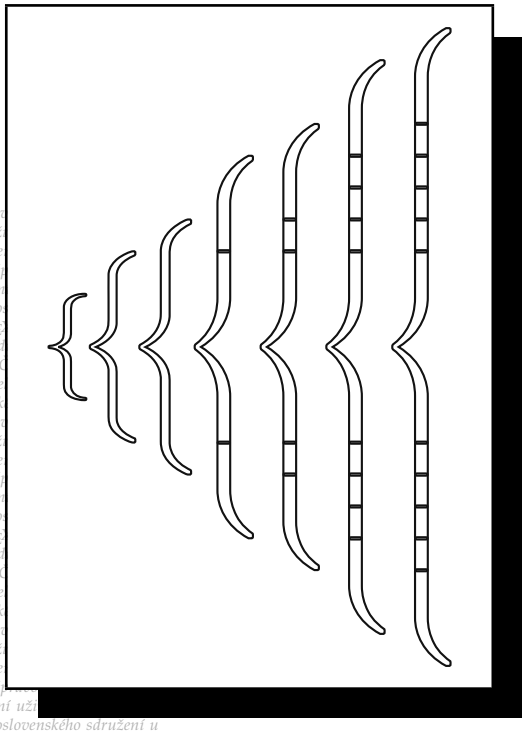
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

1–4

2016

ISSN 1211-6661 Tištěná verze

ISSN 1213-8185 Online verze

Ročník 26

OBSAH

Zpravodaj Československého sdružení uživatelů TEXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Po uplynutí dvanácti měsíců od tištěného vydání je poskytován v elektronické podobě (PDF) ve veřejně přístupném archívu dostupném přes `http://www.cstug.cz/` .

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`). Postupujte podle instrukcí, které najdete na stránce `http://bulletin.cstug.cz/` . Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí TEX Live), zejména v případě, kdy vás nelze kontaktovat e-mailem.

ISSN 1211-6661 (tištěná verze)
ISSN 1213-8185 (online verze)

Úvodník staronového předsedy

Petr Sojka

Milé čtenářky a čtenáři, příznivci kvalitní typografie sázecím systémem TEX,

otvíráte první a poslední číslo Zpravodaje v roce 2016, čtyřčíslo s články roku 2016. Dovolte mi při příležitosti mého znovuzvolení předsedou našeho zapsaného spolku malou reminiscenci a zhodnocení stavu CSTUGu a Zpravodaje.

Ohlédnutí zpět

Jelikož jsem předsedal CSTUGu kromě období 1995–2001 již poslední dvě volební období (2010–2016), je na místě malé ohlédnutí zpět, a dovolím si pár myšlenek o našem sdružení. Mnoho se od mého předání sdružení v roce 2001 kolegům Olšákovi (2001–2004) a Kubenovi (2004–2010) a za dobu mého předchozího předsedování změnilo.

Před šestnácti lety při mém předávání statutárské štafety byl CSTUG v počtu členů lokálních sdružení uživatelů (LUG) na milión obyvatel na špici pomyslného pelotonu LUG. Dnes tomu tak již není, a zájem o členství má trvale sestupnou tendenci. Naše sdružení nevzkvétá. Celosvětový TUG loni uspořádal roční kampaň s cenami za získávání nových členů. Členy se bez pobídek již nestávají běžní uživatelé TEXu prostě proto, že nemají potřebu se organizovat ve spolku uživatelů jak bylo dříve běžné. LUG se spíše stávají profesními organizacemi, kde zůstávají vesměs profesionálové TEXem se živící, nebo dlouhodobí entuziasté středního či vyššího věku.

Je oprávněné se ptát, zda CSTUG nabízí to, po čem je mezi desítkami tisíc řadových uživatelů TEXu poptávka. Je potřeba se ptát, jak plní své poslání a slouží členům. Nasnadě je otázka, zda je snižování počtu členů dáno objektivními faktory, jako je snadná dostupnost informací a TEXových distribucí na Internetu, existencí podpory TEXových guru na platformách typu StackExchange, zvýšenými možnostmi vyžití a uplatnění mimo zapsané spolky a občanská sdružení, jako je CSTUG, nebo neplněním očekávání členů od LUG.

V úvodníku staronového předsedy ve Zpravodaji 2011-1 jsem avizoval svůj minimalistický program sdružení: kromě dodržení všech zákonných povinností chodu neziskové organizace a dodržení slibů grantové finanční podpory jsem chtěl konsolidovat aspoň vydávání našeho Zpravodaje. Nechávám na čtenáři posouzení, do jaké míry se to podařilo. Nepodařilo se také řádně připomenout a oslavit čtvrtstoletí existence CSTUGu.

Sebekriticky jsem pro volby na další období hledal pro nehonorovanou a zodpovědnou funkci statutára CSTUGu svého nástupce. Přes veškeré úsilí se mi však

nikoho nového s mladickým elánem k převzetí předsednické štafety přemluvit nepodařilo. Naopak, u některých členů výboru to patrně vedlo k jejich rozhodnutí raději nekandidovat do výboru vůbec, a slíbili se dále angažovat nezávisle na závazku funkce v organizaci. Jelikož jsem ze svého předchozího dvanáctiletého vedení CSTUGu věděl, o jak náročnou a zodpovědnou funkci se jedná, s kandidaturou do výboru – a tedy s hrozbou návrhu na předsedu sdružení – jsem dlouho váhal.

Valné shromáždění

Na obvyklé přednášky před každoročním valným shromážděním se mi podařilo sehnat hned dvě přednášky svých studentů na Fakultě informatiky MU: Bc. Víta Novotného a Bc. Dávida Luptáka. Po uvedení a vyslechnutí obou přednášek (články k oběma jsou v tomto čísle) a po rozhodnutí obou přednášejících se stát aktivními členy sdružení jsem pojal důvěru v budoucnost spolku založenou na šikovných a iniciativních mladých členech. A poté, co Vít Novotný na shromáždění vyjádřil i ochotu pomoci s dosud váznoucí technickou redakcí Zpravodaje CSTUG a aktivně se nabídl přiložit ruku k dílu Honza Šustek, bylo rozhodnuto. Na následné schůzi výboru jsem s podmínkou pozitivního obratu a změn koncepce ve vydávání Zpravodaje souhlasil s opětovným zvolením do čela spolku a stal se tak staronovým předsedou sdružení.

Poděkování

Nelze než poděkovat předchozímu výboru za to, že sdružení a jeho tradice byla udržena, včetně minimalistického programu činnosti. Nejen že byly udržovány diskusní listy a zpřístupňován archiv CTAN zrcadlením do Brna, byl vytvořen nový web sdružení. Částečně byl redukován skluz ve vydávání Zpravodaje sdružení, zejména péčí jeho šéfredaktora Zdeňka Wagnera a jeho zástupce Tomáše Hály.

Setrvávající aktiva sdružení byla využívána na podporu TEXových projektů doma i ve světě. CSTUG finančně podporoval projekty fontů TEX Gyre, METAPOST, LuaTEX. Ty jsou na světě i díky finančnímu přispění CSTUGu. Díky placení kolektivního členství CSTUGu v TUGu bylo rozesíláno osm čísel časopisu TUGBoat do center TEXového života v Praze (knihovny MFF UK, FEL ČVUT a MÚ AV), Liberci, Ostravě (OU), Brně (Univerzita obrany), Bratislavě (MFF UK) a Košicích (UJŠ).

Zpravodaj

Od svého založení CSTUG vydává primárně pro komunikaci mezi svými členy Zpravodaj CSTUGu. Nedávno v tichu oslavilo periodikum čtvrtstoletí svého nepřerušeného vydávání. V předchozím období se nedařilo redakci časopis vydávat

pravidelně, jednak z důvodu nedostatku nabízených kvalitních příspěvků, jednak kvůli technické náročnosti sesazení jednotlivých čísel. Své sehrálo i pracovní vytížení protagonistů redakce, šéfredaktora a jeho zástupce, vzhledem k vysokým ambicím, které se ale včas nedařilo naplňovat a komunikovat.

Nová koncepce vydávání Zpravodaje

Vzhledem ke své pracovní zaneprázdněnosti na pozici šéfredaktora Zpravodaje rezignoval Zdeněk Wagner. Hozenou šéfredaktorskou rukavici zvedl Honza Šustek, který byl v této roli jednomyslně výborem potvrzen. Důležitý bod jeho koncepce vydávání jde vstříc autorům, kteří na redakční zpracování svých článků 'v bufferu' redakce dříve čekali nezvykle dlouho:

if máme v březnu v bufferu nemálo článků **then**
  vydáme číslo a vyprázdníme buffer
end if
if máme v červnu v bufferu nemálo článků **then**
  vydáme [jedno–dvoj]číslo a vyprázdníme buffer
end if
if máme v září v bufferu nemálo článků **then**
  vydáme [jedno–troj]číslo a vyprázdníme buffer
end if
V listopadu vydáme [jedno–čtyř]číslo a vyprázdníme buffer.

Plně se s tímto programem ztotožňuji a toto číslo připravené v rekordně krátkém čase je prvním hmatatelným důkazem jeho naplňování. Díky! Věřím, že příslib pravidelnosti přivede k zasílání článků nové autory, kteří se podělí o své zkušenosti či zpracují problematiku formou přehledového článku.

Na Valném shromáždění, jakožto nejvyšším orgánu spolku, byla tato problematika diskutována. V diskusi s přijetím role technického redaktora Zpravodaje k mé velké radosti souhlasil Vít Novotný. Že to byl dobrý krok svědčí i číslo, které držíte v rukou.[1]

Nová pravidla vydávání sice bude ještě třeba kolektivně dopracovat, zajistit transparentnost procesů redakce, pravidla recenzního řízení a otevřenost zapojení odborníků při technických výzvách sazby a přípravy Zpravodaje do tisku. Jste-li ochotni přiložit ruku k dílu recenzemi, korekturami, či svým know-how, ozvěte se prosím redakci.

[1] Příkaz `svn log|grep vnovotny|wc -l` informující o počtu revizí, které na tomto čísle od svého vstupu do CSTUGu před valným shromážděním udělal, dává tříciferné číslo. To demonstruje objem práce, který sesazení a redakce takového čísla obnáší. Případné nejednotnosti jdou na můj vrub daný tlakem číslo rychle dostat ke členům a srovnat skluz ve vydávání.

Editorial čísla

Obsah tohoto čísla začíná dvěma články v angličtině referujícími o projektech, které CSTUG podpořil finančně: TEX Gyre a SWIGLIB.

Polští kolegové podávají zevrubnou zprávu o tom, jak po dvě dekády přispívali do pokladnice volně šiřitelných fontů, včetně vývoje infrastruktury umožňující i vývoj matematických fontů a OpenType verzí fontů. Dočtete se o historii Latin Modern fontů, důležitosti kompatibility metrik fontů, vývoji rodiny fontů TEX Gyre nahrazující komerční rozšířené alternativy základních fontů Adobe, a doplňující pět rodin fontů kompletní podporou TEXové matematiky. Závěrem jsou naznačeny směry dalšího vývoje.[2]

Luigi Scarso referuje o svém projektu SWIGLIB, taktéž podpořeném finančně CSTUGem. Projekt umožňuje efektivní využití binárních knihoven v LuaTEXu, zachovávaje standardní devizy TEXových řešení, a to efektivitu a portabilitu.

Marek Pomp sdílí své zkušenosti s literárním programováním v jazyce R vhodném pro statistické výpočty a přípravu diagramů.

Vít Novotný popisuje svůj nový makrobalík Markdown umožňující stejnojmenný jazyk použít v kterýchkoli dokumentech LATEXu, plain TEXu a ConTEXtu. Poměrně rozšířený a úsporný jazyk tak nyní lze sázet kvalitním sázecím strojem, a také ve složitějších dokumentech kombinovat se složitějšími objekty nepodporovanými ve značkování Markdown. Článek demonstruje vhodnost využití dostupnosti Lua interpretu místo makroexpanze pro implementaci řešení problému parsování textu v jazyce Markdown.

Problematikou elektronických knih a jejich přípravy z klasicky značkovaných dokumentů se zabývá článek Michala Hofticha. Řešení je postaveno na konverzi do XML formátů elektronických čteček pomocí programu TEX4ht.

Neméně atraktivní a v komunitě žádanou funkcionalitou je sazba bibliografických citací vyhovujících normě ISO 690. V článku o balíku biblatex-iso690 BIBLATEXu Dávid Lupták informuje o výsledcích své bakalářské práce na Fakultě informatiky MU, která umožňuje splnit požadavky této nedávno aktualizované normy. O poptávce a významu práce svědčí rychlost, s jakou developerská komunita po jeho řešení sáhla, měřená počtem větví v repozitáři projektu.

Číslo se uzavírá překladem další části seriálu Petera Wilsona o využití LATEXových nástrojů pro cykly na práci s řetezci.

Poděkování členům, zejména kolektivním

Největší členské příspěvky pro finanční zabezpečení chodu sdružení tradičně přináší kolektivní členové, za což patří dík i jejich kontaktním osobám. Jmenovitě jde o:

1. Bílková Hana, Ústav informatiky AV ČR, v. v. i.

[2] Na výborové schůzi CSTUGu byl odsouhlasen závazek další vývoj fontů GUST E-foundry v letech 2017–2019 finančně podporovat částkou 1000 Eur ročně.

2. Bojar Štěpán, Univerzita Karlova v Praze
3. Borková Eva, Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky, Ústav matematiky
4. Buša Ján, Technická univerzita v Košiciach
5. Dvorský Jiří, VŠB-TU Ostrava, Fakulta elektrotechniky a informatiky
6. Genčev Marian, VŠB-TU Ostrava
7. Halíř Jiří, Konzervatoř, Brno
8. Jakl Ondřej, Ústav geoniky AV ČR, v. v. i.
9. Jurák Josef, Slezská univerzita v Opavě
10. Kuben Jaromír, Ministerstvo obrany ČR
11. Nemoga Karol, Matematický ústav SAV, Bratislava
12. Písačka Jan, Ústav fyziky plazmatu AV ČR, v. v. i.
13. Plch Roman, Ústav matematiky a statistiky PřF MU
14. Pospíšil Herman, Ústav makromolekulární chemie AV ČR, v. v. i.
15. Rákosník Jiří, Matematický ústav AV ČR, v. v. i.
16. Roupec Petr, Fyzikální ústav AV ČR, v. v. i.
17. Rybička Jiří, Mendelova univerzita v Brně, Ústav informačních technologií
18. Sklenák Vilém, Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky
19. Sojka Petr, Masarykova univerzita Brno, Fakulta informatiky
20. Svoboda Vojtěch, České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská
21. Šustek Jan, Ostravská univerzita
22. Vaníček Petr, Ústav teorie informace a automatizace AV ČR, v. v. i.
23. Veselý Jiří, Univerzita Karlova, Matematicko-fyzikální fakulta
24. Vyhnánek Jiří, České vysoké učení technické v Praze, Fakulta strojní, Centrum počítačových služeb

Kolektivní členové často deponují ve svých knihovnách Zpravodaj sdružení, a DVD TEXlive, neboť obojí dostávají v několika kusech, a takto přirozeně sdružují uživatele v centrech vzdělanosti na jednotlivých pracovištích. Pokud víte ve svém okolí o organizaci, která TEX používá, prosím navrhněte, aby zvážila své členství v CSTUGu.

Prioritou mého předsednictví v druhém čtvrtstoletí CSTUGu bude kromě zajištění minimálního chodu sdružení stabilizace vydávání Zpravodaje a podpora aktivních jednotlivců a projektů naplňujících poslání sdružení. S členstvím CSTUGu v CrossRef souvisí předávání metadat vydaných článků do této celosvětové databáze, a to zpětně pro články vydané od roku 1991, včetně seznamů citací. To by mohlo zvýšit viditelnost Zpravodaje a zatraktivnit i publikování ve Zpravodaji pro akademiky. Neméně důležitá bude prezentace CSTUGu na Internetu, aktuální a udržované webové stránky, funkční administrativní rozhraní databáze členů, mailové diskusní skupiny a péče o ftp archiv CTAN a sdružení.

Aktivity sdružení stojí a padají na ochotných jednotlivcích. Máte-li co sdružení nabídnout, ať už v souvislosti s výše naznačenými činnostmi sdružení, nebo byste chtěli začít novou aktivitu v duchu stanov sdružení, napište prosím na adresu výboru z tiráže!

Budeme vděčni i za pomocnou ruku v získávání nových členů, propagaci sdružení, editaci a redakci tematických webových stránek na webu sdružení, ale i za konstruktivní náměty na další aktivity či připomínky ke stávajícímu chodu sdružení.

Líbí se mi tradice německy mluvících TEXistů DANTE zvaná „Stammtische“. Přibližně jednou za měsíc se ve některých městech u jednoho stolu pravidelně scházejí zájemci o TEX a sdílejí své zkušenosti a znalosti. I to je možná příčinou, že DANTE má více členů než celosvětový TUG. Pokud byste ve svém městě chtěli tuto tradici osobních kontaktů založit, směle do toho a CSTUG jistě pomůže dát místo setkání svým členům ve známost. Jen do toho!

Summary: Introductory Word by Once and Future President

Editorial discusses CSTUG's past and future, membership issues and shaping the organization in the Internet era, together with changes related to the publishing of Zpravodaj CSTUG.

Go forth and participate in CSTUG to make the bright future of TEX & Friends a reality! *You can!*

Masarykova univerzita, Fakulta informatiky, Botanická 68a, 602 00 Brno
sojka@fi.muni.cz

GUST E-Foundry Font Projects

Bogusław Jackowski, Piotr Strzelczyk, Piotr Pianowski

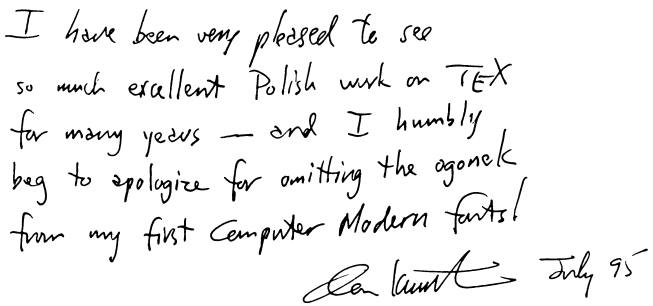
In this paper, we present a summary of our work on the TEX Gyre fonts. We also discuss inconsistencies in the existing fonts and encodings and show how the problems are solved in the TEX Gyre fonts.

Keywords: TEX Gyre fonts, Latin Modern fonts, AFDKO, GUST e-foundry.

What is a document?
It is a sequence of rectangles containing a collection of graphic elements.
What is a font?
It is a sequence of rectangles containing a collection of graphic elements.
—Marek Ryćko

1. Introduction

The Polish TEX Users Group (GUST) has paid attention to the issue of the fonts since the beginning of its existence. In a way, it was a must, because the repertoire of the diacritical characters of the *Computer Modern* family of fonts (CM), "canonical" TEX family defined as the Metafont programs (see [5]), turned out to be insufficient for the Polish language. The efforts of the GUST font team (GUST e-foundry), led by Bogusław Jackowski, were kindly acknowledged by the professor Donald E. Knuth:



Obviously, our first fonts were PK bitmap fonts programmed using Metafont. Alas, the TEX/Metafont bitmap font format never became the world-wide standard.

DOI: 10.5300/2016-1-4/7

Therefore, the next step were fonts in the PostScript Type 1 format which fairly soon became obsolescent and was replaced by the OpenType format (OTF, a joint enterprise of Microsoft and Adobe, 1996, see [31]) which is actually a common container for the Adobe PostScript Type 1 and Microsoft TrueType (TTF) formats. In 2007, Microsoft extended the OTF standard with the capability of typesetting math formulas, largely based on ideas developed for TEX, and implemented it in MS Office. Soon, TEX engines were adapted to process such math OTF fonts. Therefore our recent fonts are released in the OpenType format, which also makes them easily usable outside of the TEX realm.

So far, no OTF successor is in sight, which is both good and bad (cf. Section 7.4).

We published our partial results successively as the work progressed. This paper provides an overall summary of our work: it describes the collections of fonts prepared by the GUST e-foundry, deals with some technical issues related to the generation of fonts and their structure and puts forward a few proposals concerning future works.

This is not an overly strict report, but rather a story about our technical work on fonts, illustrated by representative examples which, we hope, show the essence of the matter. In order to keep our narration smooth, we decided not to use formal captions with explanations to figures and tables (only numbers of figures and tables are given). The relevant detailed descriptions always appear in the main text.

2. Historical background

PostScript and TEX are genetically related: their common ancestor is the ingenious idea of a programming language for the description of documents understood as a sequence of rectangular pages filled with letters and graphics. Both projects were devised nearly at the same time — at the turn of the 1970s to the 1980s.[1] And both are still alive and well, proving that the idea behind both projects was indeed brilliant.

From our perspective, the most important thing in common, and at the same time a key distinctive element, was the different handling of fonts and graphics; in other words, both systems clearly distinguished illustrations from fonts. That approach was justified by the computer technology at that time.

For both TEX and PostScript, fonts were external entities, both used metric files plus files defining glyph shapes, both defined contours as Bézier splines (planar polynomials of the 3rd degree), and for both fonts were to be prepared separately with dedicated font programs, prior to creating documents.

[1] Formally, TEX was released a little earlier — TEX in 1982, PostScript in 1984, both with earlier work.

And this exhausts the list of similarities.

TEX worked with binary metric files, TFM; its output, a device independent file (DVI), was processed by so-called drivers which made use of the "proper" fonts, that is, the relevant bitmap collections, and produced output that could be sent to a printer or to a screen. The bitmaps were prepared independently with the Metafont program(s) which interpreted scripts written in the Metafont language and generated TFM and bitmap files.

In Metafont, the shapes of glyphs are defined as Bézier curves, stroked with a "pen" and/or filled.

Basic PostScript fonts (i.e., Type 1; see [28]) employ contours defined as non-intersecting closed Bézier outlines which can only be filled.[2] The "filled outline" paradigm relates also to the Microsoft TTF format and, thereby, to the OTF format.

PostScript Type 1 fonts are usually (but not necessarily) accompanied by corresponding ASCII metric files (AFM), not used by the interpreters of the PostScript language. In the Microsoft Windows operating system, making an already complex situation even more complex, binary printer font metric files (PFM) were introduced for Windows drivers that used PostScript Type 1 fonts.

For a long time, only commercial programs for generating PostScript Type 1 fonts were available. Only in 2001, George Williams released his remarkable FontForge program (initially dubbed PfaEdit; [25]). FontForge can generate outline fonts in many formats, including PostScript Type 1 and OTF.

PostScript was promptly (and rightly) hailed as *the* standard for printers and, more importantly, for phototypesetters, therefore a driver converting DVI files to PostScript became necessary. Fortunately for TEXies, PostScript is equipped also with Type 3 fonts; glyphs in Type 3 fonts can be represented by nearly arbitrary graphic objects, in particular, by bitmaps, therefore the making of a PostScript driver for converting DVI files to PostScript was possible already in 1986, when Dvips, the first and still most popular driver was released by Tomas Rokicki. (It's a pity that the idea of Type 3 fonts was not supported and developed by Adobe.)

There were a few unsuccessful attempts to convert the basic TEX font collection, CM, to the PostScript Type 1 format automatically, thus preserving the parameterization. The main hindrance was the excessive usage of stroked (both painted and erased) elements in the CM font programs, while, as was mentioned previously, the PostScript Type 1 and OTF formats accept only filled shapes.

The "filled outline" paradigm was a convenient optimization at the beginning of the computer typesetting era, when, for example, the generating of the complete

[2] The PostScript Type 1 documentation [28], p. 34, mentions the possibility of stroking: *a Type 1 font program can also be stroked along its outline when the user changes the PaintType entry in the font dictionary to 2. In this case, overlapping subpaths will be visible in the output; this yields undesirable visual results in outlined characters.* In practice, this possibility is not used.

collection of bitmaps for the CM fonts at resolution, say, 240 dots per inch (typical for dot matrix printers) took a few days. Nowadays, the paradigm still thrives by virtue of tradition: there is an abundance of such fonts and, and what is worse, all operating systems support only this kind of font.

3. First steps

Taking the above into account, we made up our minds to design our own programmable system for generating fonts in "world-compatible" formats.

3.1. Our tools

Our primary tool was MetaPost [4], a successor of Metafont, which promised well as a tool for making PostScript Type 1 fonts due to its native PostScript output. We called our MetaPost-based package MetaType 1 [13]. It was instantiated as a set of scripts using, besides MetaPost, T1utils, that is, Lee Hetherington's (dis)assembler for PostScript Type 1 fonts (cf. [3, 4]). A few scripts written in Gawk and Perl were also employed.

On the one hand, such a simple approach turned out to be insufficient for generating OTF fonts, in particular OTF math fonts. On the other hand, it turned out to be flexible enough to include an extra external step for making OTF fonts. For text fonts, we employed the *Adobe Font Development Kit for OpenType* (AFDKO [26]); for math fonts, the FontForge library governed by Python scripts [15, 25]. In the future, we want replace AFDKO by a FontForge+Python utility (cf. Section 7).

A set of MetaPost macros in the MetaType 1 package defines two important procedures, essential for generating non-intersecting outlines and heavily used in our font programs: finding a common outline for overlapping figures, known also as removing overlaps, and finding the outline of a pen stroke, known as expanding strokes or finding the pen envelope.

Another important feature, hinting, is implemented, but, in the end, we decided to avoid manual hinting, since it is difficult to control and yields mediocre results. Metafont has no notion of hinting — the Metafont language simply offers rounding. Moreover, the language for describing outlines in PostScript Type 1 fonts cannot express even as trivial a mathematical operation as rounding.

For low-resolution devices, controlled rounding is crucial — hence the idea of "hinting", that is, controlled rounding. Alas, hinting algorithms remain undisclosed, especially with regard to commercial typesetting devices such as phototypesetters. One can presume, however, that low-resolution devices are bound to disappear sooner rather than later: the resolution of display devices has reached almost 600 dpi and 1200 dpi (and more) for printers is nowadays nothing special.

Therefore, running with the hare and hunting with the hounds, we decided to hint our recently released OTF fonts automatically with FontForge.

3.2. Trying our tools out

We tested our newborn MetaType 1 engine against a simple example, namely, Donald E. Knuth's `logo` font [17]: the sources, originally written in the Metafont language, were adapted to MetaType 1's requirements. The distributed package contains both MetaType 1 sources and the resulting PostScript Type 1 files for the `logo` font [13].

The test proved the usefulness of the approach; hence, in 1999, we started a larger project: the programming of the long-established Polish typeface *Antykwa Półtawskiego* as a parameterized font. The preliminary family of fonts was released in 2000. Ten years later, prompted by the TEX community, we released the enhanced version of *Antykwa Półtawskiego* with the relevant OTF files [7]. In parallel, Janusz M. Nowacki used MetaType 1 to generate several replicas of Polish fonts, namely, *Antykwa Toruńska*, *Kurier*, *Iwona*, and *Cyklop* [22].

4. Latin Modern collection of fonts

Recall that the repertoire of diacritical characters in CM fonts was insufficient for most languages using diacritical characters. The TEX accenting mechanism (the `\accent` primitive), meant as a solution to this problem, was unsatisfactory — for example, accents and hyphenation conflicted. The problem was recognized relatively early and various approaches were used to remedy the situation.

For example, the Polish extension of CM in the PK format was prepared in Poland (PL fonts, [8]), but this worked only for Polish TEXies.

Also worthy of note is the *European Computer Modern Fonts* (EC) project led by Jörg Knappen and Norbert Schwarz, triggered during the TEX Users Group conference in Cork, 1990, and finished in 1997 [16]. The EC metric files, however, are slightly incompatible with CM metric files, by circa 0.025%.[3]

A rather general technique was applied by Lars Engebretsen, who attempted to eliminate the necessity of using the `\accent` primitive by making virtual fonts, dubbed *Almost European* (AE), containing quite a large set of the European diacritical characters [1].

Hyphenation worked with AE fonts, though still unsatisfactorily — for example, coinciding of such accents as cedilla or ogonek with a main glyph is inconvenient

[3]The reason behind this discrepancy is a peculiarity of Metafont arithmetic: the formula $1/36 * i$ yields different results than the formulas $i/36$ and $1/36i$; for example, for $i = 3600$ the results are 99.97559 for the first formula and 100 for the latter formulas. The first formula was used in the EC fonts (in the `gendef` macro).

when a non-intersecting outline is required (for cutting plotters, for outlined titles, and so on). Moreover, the virtual fonts are obviously unusable outside the TEX realm.

4.1. Latin Modern fonts in the PostScript Type1 format

In 2002, during the EuroBachoTEX meeting, a proposal of converting Engebretsen's AE fonts into the PostScript Type 1 format and augmenting with them the set of necessary diacritical characters was put forward by representatives of European TEX user groups. We had no choice but to accept the proposal with delight.

Our initial plan was to use the AE fonts as our departure point; we even wanted to preserve the original name, *Almost European*, coined by Lars Engebretsen. It turned out, however, to be much more efficient to prepare the enhanced version of the CM fonts from scratch, and so sticking to Lars Engebretsen's name seemed inadequate, because the differences were too essential.

All in all, inspired by both EC and AE fonts, we came up with the *Latin Modern* (LM; see [11]) project which was accepted by the user groups.

Fortunately for us, freely available quality CM fonts in the PostScript Type 1 format already existed. In the 1980s and 90s, they were produced (from traced bitmaps improved by very solicitous manual tuning) for commercial purposes by Blue Sky Research and Y&Y. Nearly a decade later, they were released to the public thanks to the efforts of the American Mathematical Society.[4]

We converted the PostScript Type 1 files of the CM fonts to MetaType 1 and wrote the MetaPost software relevant for generating the characters we decided to add (mainly diacritical letters). The work had already been partially done by Janusz M. Nowacki, who prepared the PostScript Type 1 version of the PL fonts in 1997. The official version of the LM fonts, 1.000, was eventually released in 2006 (in the meantime, several unofficial versions were released for testing purposes). The LM collection of fonts consisted of 72 text fonts, each counting about 700 glyphs, plus 20 CM-like math fonts.

In 2009, an extensive revision of the LM fonts was carried out: the text fonts now contain more than 800 glyphs each (altogether more than 60,000 glyphs) and the glyphs conform to the changes introduced by Donald E. Knuth in 1992.

Almost all of the CM text fonts have counterparts in the LM family; the exceptions are one monospaced font, `cmtex10`, emulating Donald E. Knuth's keyboard layout, and the rarely used `cmff10`, `cmfi10`, `cmfib8`, and `cminch`. So

[4] In 1997, a consortium of scientific publishers (American Mathematical Society, Elsevier Science, IBM Corporation, Society for Industrial and Applied Mathematics, and Springer-Verlag) in cooperation with Blue Sky Research and Y&Y decided to release these excellent fonts non-commercially; in order to assure the authenticity of the fonts, copyright was assigned to the American Mathematical Society. (`http://www.ams.org/publications/type1-fonts`).

far, nobody has complained about this inconsistency. Instead, encouraged by Hans Hagen, we decided to create 10 variants of typewriter LM fonts not having counterparts in the CM family: `lmtlc10`, `lmtk10`, `lmtl10`, `lmvtk10`, and `lmvtl10` (monospaced light condensed and monospaced and variable-width dark and light, respectively) plus their oblique variants `lmtko10`, `lmtlo10`, `lmtlco10`, `lmvtko10`, and `lmvtlo10`.

4.2. Latin Modern fonts in the OTF format

It was relatively easy to prepare the LM family of fonts in the OTF format using the AFDKO package: it mainly necessitated preparing a few extra data files in the *OpenType Feature File Specification* language [32]. Needless to say, the experience gathered at this stage came in handy during the work on the TG fonts (see Section 5).

There was trouble, however, with the 20 math fonts. We provided the respective LM equivalents in PostScript Type 1 format. For compatibility with the (obsolete) PL fonts, the symbol fonts, `lmsy*` and `lmbsy*`, contain two extra glyphs: slanted greater-or-equal and less-or-equal signs, used traditionally in Polish math typography. As the math extension for OpenType did not exist yet, we decided not to convert these fonts to OTF. We knew that the companies that had invented and maintained the OTF standard, in cooperation with the American Mathematical Society and the Unicode Consortium, were working on extending the standard with math typesetting capabilities. We expected that by using the enhanced OTF specification we would be able to create a TEX-compatible math OTF collection. Alas, the Unicode Consortium report on Unicode support for mathematics [37], followed by the initially confidential Microsoft specification [29], snuffed out our hopes. It turned out that OTF math and TEX math cannot be reconciled. More information on the interrelationships between OTF and TEX math can be found in Ulrik Vieth's thorough analysis [24].

4.3. Repertoire issues

Our primary aim was to provide a repertoire of diacritical letters rich enough to cover all European languages. We thoroughly exploited Michael Everson's comprehensive study of European alphabets [2], as well as other sources. Several other languages using Latin-based alphabets, such as Vietnamese, Navajo and Pali, are covered.

Initially, contrary to the *Latin Modern* name, we considered including Cyrillic alphabets also. Having thought the matter over, we decided, with regret, to abandon this idea and concentrated our efforts on OTF math fonts.

Besides diacritical characters, the Latin Modern fonts contain also a number of glyphs traditionally present in TEX fonts, such as Greek symbols, currency

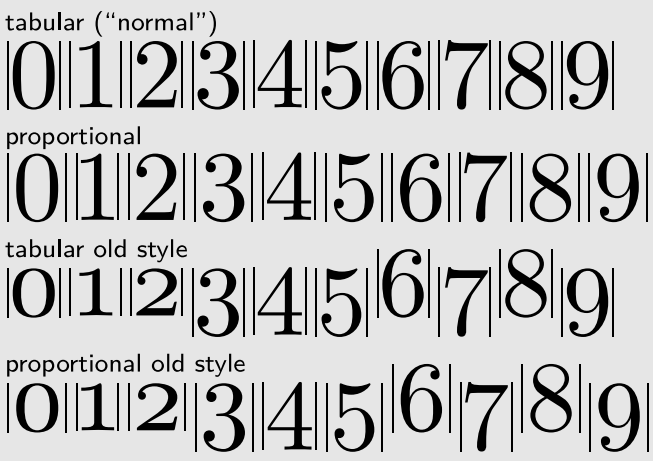


Figure 1

symbols, technical symbols, etc. Detailed description of the contents of the fonts can be found in the document entitled *The Latin Modern Family of Fonts. Technical Documentation*, included in the LM distribution package [11].

Two groups of glyphs are widely used in typography but neglected to a certain extent in the CM fonts, namely, *caps and small caps* and *old style numerals*, also known as *text figures* or *nautical digits*; the latter name originates from their widespread use in tables in nautical almanacs at one time. For reasons hard to explain, the caps and small caps were implemented in the CM family as a separate font, while the old style numerals (upright!) are in the math italic font (`cmmi*`).

The LM fonts incorporated caps and small caps from the CM family, together with its width idiosyncrasy: the `lmcsc10` font, like `cmcsc10`, has capital letters wider than the `lmr10` font by circa 8%. There are two caps and small caps fonts in the LM collection, namely, the regular and typewriter specimen, `lmcsc10` and `lmtcsc10`, as in CM, plus their oblique variants, `lmcsco10` and `lmtcsco10`, absent from CM. In principle, small caps glyphs could be transferred to other fonts, but we decided to not alter the framing of the original CM family, more so as CM has no sans-serif caps and small caps; however, the problem of extending the LM family with bold counterparts of `lmcsc10` and `lmtcsc10` (and their oblique variants), raised repeatedly by CM/LM users, needs serious consideration.

Concerning old style numerals, we could not accept the CM oddity and included them in all text fonts of the LM family. Further, all numerals come in 2 ‘flavors’: normal (fixed-width a.k.a. tabular) and proportional (variable-width, having balanced sidebearings) which altogether yields 4 variants — see Figure 1.

The TFM format contains only 256 slots for glyphs, thus, the whole repertoire of glyphs cannot be accessed at once if TEX is used in a “traditional” way, that

is, with TFM files. In particular, accessing the different kinds of numerals when using PostScript Type 1 fonts plus TFM metrics turns out to be clumsy; as a result, only tabular old style numerals are available in our package, in the TS1 encoding (see below). On the one hand, OTF fonts seem more convenient as they do not impose such a restriction; for example, all numerals can be accessed by using the OTF feature mechanism, more precisely, by the features `onum`, `lnum`, `pnum`, and `tnum` [33]. On the other hand, OTF metric data cannot, in general, be fully compatible with TFM metric data because the glyph widths in OTF fonts must be represented by integer quantities. This should be considered a drawback by TEXies — see Section 4.4.

Following the LATEX tradition, we provided several encodings for the LM fonts, namely:

- ◇ CS (CS TUG) encoding (`cs-*.tfm`),
- ◇ EC (Cork) encoding (`ec-*.tfm`),
- ◇ L7X (Lithuanian) encoding (`l7x-*.tfm`),
- ◇ QX (GUST) encoding (`qx-*.tfm`),
- ◇ RM ("regular math", used in OT1 and OT4) encodings (`rm-*.tfm`),
- ◇ Y&Y's TEX'n'ANSI a.k.a. LY1 encoding (`texnansi-*.tfm`),
- ◇ T5 (Vietnamese) encoding (`t5-*.tfm`),
- ◇ Text Companion for EC fonts a.k.a. TS1 (`ts1-*.tfm`).

The LATEX support for all these encodings, due to Marcin Woliński, is also part of the LM distribution.

TFM files nominally representing the same encoding do not always define the same set of characters; for example, the character sets of `cmr10.tfm` and `cmtt10.tfm` differ. The original CM fonts comprise 7 different character sets, with an idiosyncratic difference between the `cmr10` and `cmr5` layouts. As a remnant of the CM design, there are 5 different character sets of the LM text fonts:

1. 821 glyphs (basic set): `lmb10 lmbo10 lmbx10 lmbx12 lmbx5 lmbx6 lmbx7 lmbx8 lmbx9 lmbxi10 lmbxo10 lmdunh10 lmduno10 lmr10 lmr12 lmr17 lmr5 lmr6 lmr7 lmr8 lmr9 lmri10 lmri12 lmri7 lmri8 lmri9 lmro10 lmro12 lmro17 lmro8 lmro9 lmss10 lmss12 lmss17 lmss8 lmss9 lmssbo10 lmssbx10 lmssdc10 lmssdo10 lmsso10 lmsso12 lmsso17 lmsso8 lmsso9 lmu10 lmvtk10 lmvtko10 lmvtl10 lmvtlo10 lmvtt10 lmvtto10`
2. 824 glyphs: `lmssq8 lmssqbo8 lmssqbx8 lmssqo8`
   extra characters: `varI varIJ varIogonek`
3. 814 glyphs: `lmcsc10 lmcsco10`
   missing characters: `f_k ff ffi ffl fi fl longs`
4. 785 glyphs: `lmtk10 lmtko10 lmtl10 lmtlc10 lmtlco10 lmtlo10 lmtt10 lmtt12 lmtt8 lmtt9 lmtti10 lmtto10`
   missing characters: `f_k ff ffi ffl fi fl Germandbls hyphen.prop IJ ij`

`permyriad servicemark suppress trademark varcopyright varregistered zero.oldstyle zero.prop one.oldstyle one.prop two.oldstyle two.prop three.oldstyle three.prop four.oldstyle four.prop five.oldstyle five.prop six.oldstyle six.prop seven.oldstyle seven.prop eight.oldstyle eight.prop nine.oldstyle nine.prop`

5. 784 glyphs: `lmtcsc10 lmtcso10`
   missing characters: as in 4, also `longs`

4.4. Compatibility issues

We did our best to provide outline fonts that can be used as a replacement for CM fonts. To a certain extent, we managed to achieve this goal, namely, the PostScript drivers which process TEX documents typeset with CM metric files, can use either CM or LM PostScript Type 1 fonts — special map files for PostScript Type 1 fonts are available for this purpose. The metric files, however, cannot be used replaceably, because the typesetting algorithms are intrinsically unstable — even tiny (rounding) errors may yield glaringly different results.

Therefore, LM users also cannot expect PostScript Type 1 and OTF fonts to be used replaceably. Recall that the OTF format requires integer number representation for glyph widths. The "reference" quantity is the *em* unit: 1 *em* = 2048 units for fonts using splines of the 2nd degree, 1 *em* = 1000 units for fonts using splines of the 3rd degree (e.g., our LM and TG fonts). Therefore, in our case, the difference in width is on average 1/2000 *em* (twice as large as the variation in the EC widths), that is, circa 0.005 *pt* for 10-point fonts.

Because the MetaType 1 sources of the LM fonts are the result of conversion from PostScript Type 1, the widths stored in the LM TFM files are not identical to the respective original CM widths. They are closer, however, to the original quantities by an order of magnitude compared to the EC and OTF widths. At the cost of great effort (by referring to the Metafont sources), we might have eliminated rounding errors in LM widths. But it would not cure the problem of (non-)replaceability, as widths are not the only source of trouble. Differences in heights and depths of glyphs may also yield unexpected behavior of the TEX typesetting algorithm.

The problem of heights and depths in TEX turns out to be unavoidable, and quite serious: the TFM format permits by design only 16 different heights and depths, including the obligatory entries containing the value zero. If there are in fact more heights and depths in a given font, their number is cleverly reduced to 16 by Metafont (as well as by the MetaPost and TFtoPL programs). One of the certainly unwanted results is that the same glyph in different encodings may have different heights and/or depths! For example, the height of the letter 'A' is 6.88875 pt in the `rm-lmr10.tfm` file (this layout is an extension to 256 slots of

the cmr10 layout), it is 6.99648 pt in the t5-lmr10.tfm file (Vietnamese layout), while in the canonical cmr10.tfm file it is 6.83331 pt.

This is not the end of the list of possible sources of incompatibility between CM and LM fonts. Positioning of the accents is also a long story. These and related aspects are explained minutely in [12].

Finally, let us consider a somewhat atypical example of incompatibility between the LM and CM fonts related to Donald E. Knuth's mistake in a CM ligtable program, uncorrectable for obvious reasons but basically harmless; namely, roman.mf contains the following:

```
% three degrees of kerning
k#:=-.5u#; kk#:=-1.5u#; kkk#:=-2u#;
ligtable "k":
  if serifs: "v": "a" kern -u#, fi
  "w": "e" kern k#, "a" kern k#,
       "o" kern k#, "c" kern k#;
```

The culprit is the if serifs clause: the kern pair 'ka' appears twice in the TFM files of serif fonts with the values $-u\#$ and $-0.5u\#$, respectively, as is easily seen in the following fragment of the cmr10.pl file (the respective lines are marked with arrows):

```
(CHARACTER C k
   (CHARWD R 0.527781)
   (CHARHT R 0.694445)
   (COMMENT
      (KRN C a R -0.055555)⇐
      (KRN C e R -0.027779)
      (KRN C a R -.027779) ⇐
      (KRN C o R -0.027779)
      (KRN C c R -0.027779)
      )
   )
```

Moreover, there are no 'va, 'vc', 've', and 'vo' kern pairs in sans-serif fonts, although there are 'kc', 'ka', 'ke', 'ko', 'wa', 'wc', 'we', and 'wo' kern pairs in these fonts. We could not see the reason for ignoring 'v' in this context, thus we decided to add the relevant kern pairs in the LM fonts; we also added quite a few other kern pairs missing, in our opinion, from the CM fonts, for example, 'eV' and 'kV'.

Summing up, we believed that we had good reasons for giving up the struggle for a "100-percent compatibility" between LM and CM metrics, whatever that would mean, and to confine ourselves to providing the mentioned replaceability of outlines.

5. The TEX Gyre collection of fonts

Heartened by the results of the LM enterprise, we accepted without hesitation the next proposal: the "LMization" of the family of fonts provided by Ghostscript as a replacement for the renowned *Adobe base 35* fonts, generously released by the URW++ company under free software licenses.

- ◇ ITC Avant Garde Gothic (book, book oblique, demi, demi oblique)
- ◇ ITC Bookman (light, light italic, demi, demi italic)
- ◇ Courier (regular, regular oblique, bold, bold oblique)
- ◇ Helvetica (medium, medium oblique, bold, bold oblique)
- ◇ Helvetica Condensed (medium, medium oblique, bold, bold oblique)
- ◇ New Century Schoolbook (roman, roman italic, bold, bold italic)
- ◇ Palatino (regular, regular italic, bold, bold italic)
- ◇ Symbol
- ◇ Times (regular, regular italic, bold, bold italic)
- ◇ ITC Zapf Chancery (medium italic)
- ◇ ITC Zapf Dingbats

Since our aim was "LMization", we excluded the Symbol and ITC Zapf Dingbats non-text fonts from the scope of our interest.

After a brief (but heated) debate, the name of the project and of its constituent fonts were coined. The project was dubbed TEX Gyre (TG) and the following names were accepted (the respective file name kernels, original Adobe names and Ghostscript, that is, URW, names are given in parentheses):

- ◇ TG Adventor (`qag` / ITC Avant Garde Gothic / URW Gothic L)
- ◇ TG Gyre Bonum (`qbk` / ITC Bookman / URW Bookman L)
- ◇ TG Cursor (`qcr` / Courier / Nimbus Mono L)
- ◇ TG Heros (`qhv` / Helvetica / Nimbus Sans L)
- ◇ TG Heros Condensed (`qhvc` / Helvetica Condensed / Nimbus Sans L Condensed)
- ◇ TG Schola (`qcs` / New Century Schoolbook / Century Schoolbook L)
- ◇ TG Pagella (`qpl` / Palatino / URW Palladio L)
- ◇ TG Termes (`qtm` / Times / Nimbus Roman No9 L)
- ◇ TG Chorus (`qzc` / ITC Zapf Chancery / URW Chancery L)

We initially considered including Cyrillic alphabets, but, as in the case of the LM fonts, we eventually abandoned this idea, also with regret, the more so as the Ghostscript fonts at that time contained an (apparently unfinished) set of Cyrillic glyphs.

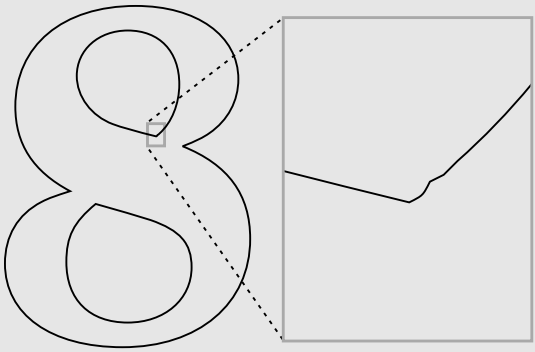


Figure 2

We expected that the main effort would be the making of extra glyphs plus maybe correcting outlines here and there. To our surprise, quite a few glyphs required tuning because of evident errors in outlines. One of the most striking examples is the glyph 'eight' from the URW Schoolbook bold font[5] — see Figure 2.

Mostly, we removed redundant or wrong nodes (points) from the outline definitions, deleting more than 5% of them in all. In the case of TG Pagella, however, the insertion of extra nodes turned out necessary. The chart in Figure 3 shows some statistics for the upright TG fonts. The diagram concerns the version of the Ghostscript fonts which we used as our starting point. We used circa 350 glyphs from each font. The total number of nodes in these glyphs varied from circa 10,000 to 25,000 per font. In the current release, the TG text fonts count almost 1100 glyphs each with the number of nodes varying from circa 30,000 to 65,000 (for sans-serif and serif italic variants, respectively; see [14]).

5.1. Repertoire issues

The difference in the number of glyphs between the TG and LM text fonts (the former having circa 250 glyphs more per font) is due mainly to the presence of small caps in the TG fonts (225 glyphs per font). Also unlike the LM fonts, each TG font contains the complete Greek alphabet and a few technical glyphs, such as 'lozenge' and 'lscript'.[6] Except for these, the LM and TG fonts share the same repertoire of glyphs and the same set of TFM encodings (see Section 4.3).

The LaTeX support for these encodings was also provided by Marcin Woliński.

[5] Recently, the font has been renamed to 'C059 bold'; the bug was removed from the Ghostscript distribution only in 2015, although the TeX Collection 2016 distribution still contains (due to the legacy reasons) the faulty glyph.

[6] For historical reasons, the LM fonts contain a few additional variants of the base, left, and right double quotes, absent from the TG fonts.

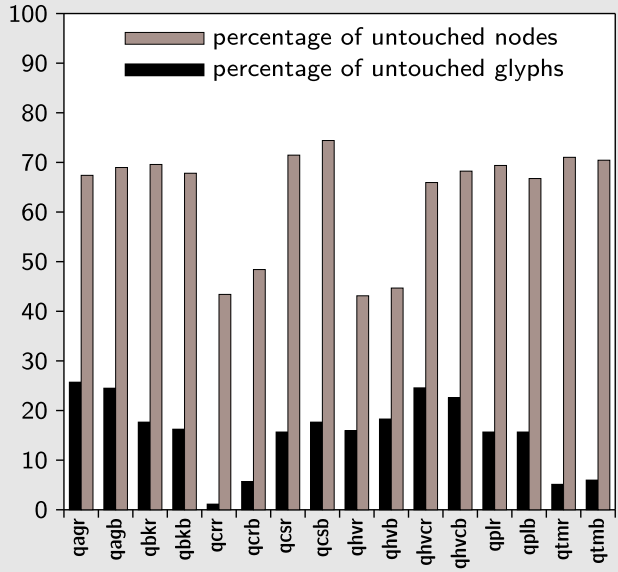


Figure 3

5.2. Compatibility issues

The consistency of the widths of the original Adobe and the respective TG glyphs was one of our main concerns, as the TG fonts were meant as potential replacements for the Adobe fonts. It turned out, however, that the original font metric files [27], contained apparent metric flaws which we decided, with some hesitation, not to retain.

A typical example (concerning Helvetica, a.k.a. Nimbus Sans L, a.k.a. TG Heros) is depicted in Figure 4. Both Spanish '¡' and Scandinavian 'ø' glyphs belong to the Adobe Standard Encoding set, hence one can expect that they should be considered important and thus unchangeable. Nevertheless, we could not see a reason for using widths different from the '!' and 'o' widths, respectively, and certainly there is no substantiation for the asymmetry of sidebearings, especially in the case of 'ø'; therefore, we decided to alter the metrics.

Fortunately, there are few such cases in the TG collection; each is mentioned in the documentation of the TG fonts [14].

Because the original widths for the TG fonts, unlike in the LM collection, were integer numbers, there is no metric discrepancy between the PostScript Type 1 and OTF font formats, as far as widths are concerned. Heights and depths, however, are subject to the same restrictions as discussed in Section 4.4.

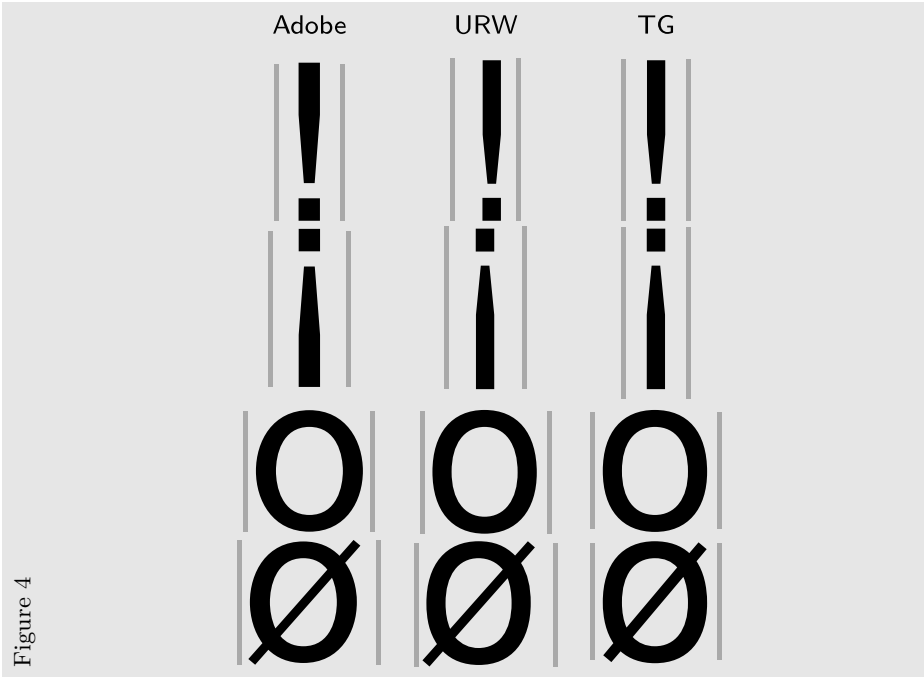


Figure 4

6. OTF math fonts

The chronic problem of lack of math support for the TG collection became the impetus for our third venture: math fonts for the LM and TG collections in the OTF format. The recent update of the LM and TG fonts took place at the end of 2009. The math extension for the OTF format ([29]) had been released and there existed the FontForge font editor ([25]) capable of generating such fonts. So we embarked upon an expedition into unknown regions — since then we have focused our attention on the work on OTF math fonts, again with the benevolent encouragement and support from the TEX users groups.

As we should have expected, the task turned out interesting and absorbing, and, according to Hofstadter's Law,[7] we spent more time on it than we expected. From the very beginning, we aimed at making a collection of mutually consistent math OTF fonts and we underestimated the heterogeneity of the sources of additional alphabets and the problem of interrelationships — works on subsequent fonts entailed moving backwards to the fonts which we had prematurely considered

[7] Hofstadter's Law: *It always takes longer than you expect, even when you take into account Hofstadter's Law* — Douglas R. Hofstadter.

ready. Nevertheless, in 2011, we happily announced the release of our first math OTF font, namely, Latin Modern Math. Altogether, six math fonts have been released by the GUST e-foundry so far [9, 10]:

- ◇ TG Latin Modern Math
- ◇ TG Bonum Math
- ◇ TG Schola Math
- ◇ TG Pagella Math
- ◇ TG Termes Math
- ◇ TG DejaVu Math

This amounts to nearly half of all OTF math fonts released in the world. Besides these, the following OTF math fonts have been released: Asana by Apostolos Syropoulos, Neo-Euler and XITS by Khaled Hosny, STIX by the STI Pub companies,[8] Cambria Math by Microsoft,[9] Lucida Math by Bigelow & Holmes, and Minion Math by Johannes Küster; the latter three fonts are distributed commercially.

6.1. OTF math font contents

Math OTF fonts, as we expounded in [10], are truly nasty beasts. In accordance with [35] and [37], they are expected to contain a plethora of glyphs: letters, arrows, math operators and delimiters, geometrical shapes, technical symbols, etc. The presence of some of them, particularly the (over)abundance of peculiar geometrical shapes and arrows, is hard to substantiate in our opinion.

Initially, we planned also releasing the math companion to the TEX Gyre sans-serif fonts, TG Adventor and TG Heros, but the Unicode specification for the contents of math fonts, [37], turned out definitely "serif-oriented". Let us take, for example, the arrangement of the LM Math font shown in Table 1: following the cited specification, we combined several LM source fonts into a single complex font. As the table clearly shows, the basic subsets, that is, plain, bold, italic and bold italic are assumed to consist of serif glyphs by default. It is not obvious how the table should be adjusted to suit sans-serif math fonts. Work on this issue is in progress.

The original CM fonts, and thus the LM fonts, do not contain the complete sans-serif Greek. We generated the missing glyphs using modified Metafont sources in order to generate outlines instead of bitmaps and tuning the result manually, if required.

[8] The STIX project began through the joint efforts of American Mathematical Society (AMS), American Institute of Physics Publishing (AIP), American Physical Society (APS), American Chemical Society (ACS), Institute of Electrical and Electronic Engineers (IEEE), and Elsevier Science; these companies are collectively known as the STI Pub companies.

[9] Cambria Math was the first math font published, conforming to the specification *MATH — The mathematical typesetting table* [29]. It was released by Microsoft in 2007, along with a MS Office version equipped with the capability of handling the math font and editing math formulas.

category	charset	source fonts
plain (upright, serif)	L*, G, D	`lmr`, `lmmi` (upright)
bold	L, G, D	`lmbx`, `lmmib` (upright)
italic	L, G	`lmmi`
bold italic	L, G	`lmmib`
sans-serif	L, D	`lmss`
sans-serif bold	L, G, D	`lmssbx`
sans-serif italic	L	`lmsso`
sans-serif bold italic	L, G	`lmssbo`
calligraphic	L	`eusm` (slanted)
bold calligraphic	L	`eusb` (slanted)
Fraktur	L	`eufm`
bold Fraktur	L	`eufb`
double-struck	L, D	`bbold` (by Alan Jeffrey)
monospace	L, D	`lmtt`

L, G, D — Latin, Greek and digits, respectively
L* — contains also diacritical letters and punctuation

Table 1 *All the alphanumeric glyphs specified in the table, except for the "plain" ones (first row), are given special mathematical Unicode slots — see [37]*

Moreover, a math font is bound to contain many other characters, most notably glyphs used for subscripts of the 1st and 2nd order, used also for superscripts; we'll refer to them shortly *pars pro toto* as subscripts. They are accessed by the OTF feature mechanism, more precisely by the math extension feature `ssty` [33].

Extensible symbols, like large brackets or radicals, are another important group of math-oriented glyphs. An extensible symbol consists of a collection of a few components (the left part of Figure 5) assembled by the typesetting engine into a seemingly single character (the right part of Figure 5).

First, a glyph of adequate size is searched for in the so-called chain of glyph variants (here: the three leftmost curly braces). If a proper glyph is not found, the typesetting engine assembles a respectively large symbol from the relevant pieces using a fairly complex algorithm — everybody who has attempted unsuccessfully to fit braces around a formula according to one's desire probably knows it.

In the case of the radical symbol, the situation is still more complex, because the OTF and TEX geometric structure, as well as the relevant metric data of the components differ, as is shown in Figure 6 which visualises "stages" of the assembling of an extensible radical symbol.

In both cases, the radical symbol is assembled from glyphs taken from a relevant font and a line (rule) drawn by the typesetting program at the top of

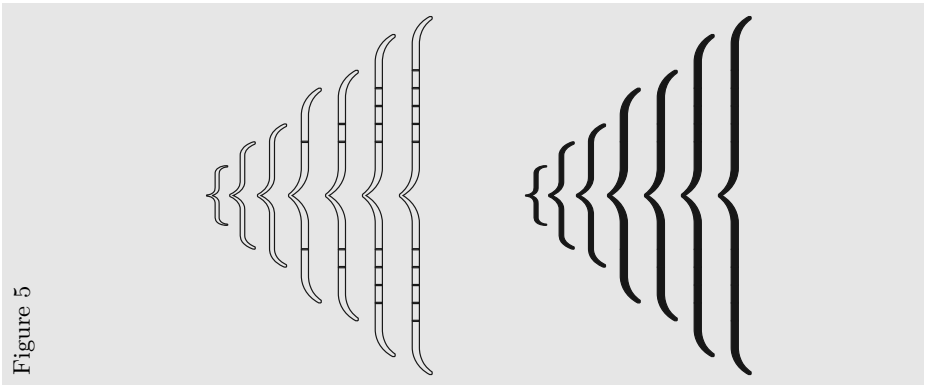


Figure 5

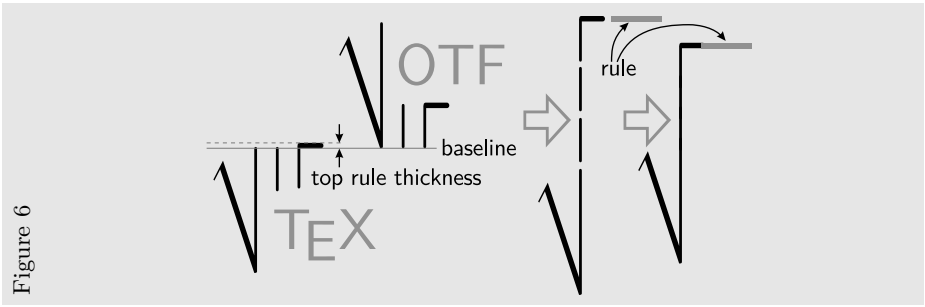


Figure 6

the symbol (marked with a gray color). The thickness of the rule, however, is given explicitly as a parameter in math OTF files, while it is inferred from the height of the top element of the radical symbol in TEX (recall the problem of the limited number of heights in a TEX metric file).

There is an important difference between the TEX and OTF math font specification concerning extensible glyphs: TEX is equipped with the ability to assemble compound glyphs from pieces only vertically, while the OTF format offers both horizontal and vertical assembling. In TEX (i.e., in the `plain` TEX format), such glyphs as horizontal braces are defined by special macros using rules and a few glyphs from a relevant font:

```
\def\downbracefill
  {$\m@th \setbox\z@\hbox{$\braceld$}%
  \braceld\leaders\vrule height\ht\z@
     depth\z@\hfill\braceru
  \bracelu\leaders\vrule height\ht\z@
     depth\z@\hfill\bracerd$}
\def\upbracefill [...]
```

Incidentally, it seems that diagonal extensible glyphs have not yet been invented. Could it be that the conundrum is too difficult to solve?

More on the differences between the TEX and OTF specifications of the structure of math fonts can be found in Ulrik Vieth's survey [24].

In the case of LM Math, we were fortunate to have well-known clean sources for a base, already containing 7-point and 5-point variants, suitable for typesetting subscripts, and components for assembling extensible glyphs.

We have to confess, however, that we were not especially delighted with the Computer Modern calligraphic script. More pleasingly designed, to our eyes, are the calligraphic letters of the renowned Euler family. Therefore, we decided to transfer the glyphs from the Euler fonts (slanting them slightly) to the LM Math font:

In the case of the TG math fonts, the situation was slightly worse. The basic sources, that is, the text fonts, were already improved by us, but the sources of the relevant additional character sets were highly heterogeneous. We used freely available fonts of the best possible quality as our base; nevertheless, much manual tuning was necessary (recall the tuning of the sources of the TG text fonts).

Moreover, not all suitable fonts had acceptable free licenses. In a few cases, we had to contact the authors personally. It should be emphasized that in all cases the authors, if we managed to reach them, courteously agreed to make their fonts available for our purposes. The following external fonts were used in the TG math fonts project (in alphabetical order):

- ◇ Lato by Łukasz Dziedzic (TG Schola Math)
- ◇ Kerkis by Apostolos Syropoulos and Antonis Tsolomitis (TG Bonum Math)
- ◇ Leipziger Fraktur replica by Peter Wiegel (TG Bonum Math and TG Termes Math)
- ◇ Math Pazo by Diego Puga (TG Pagella Math)
- ◇ Odstemplik by Grzegorz Luk (gluk) (TG Pagella Math)
- ◇ Theano Modern by Alexey Kryukov (TG Schola Math)

We are most grateful to all the authors for their prominent aid.

6.2. Visual issues

Observe that the same monospaced alphanumeric characters (excerpted from TG Cursor) are shared by TG Bonum Math, TG Schola Math, and TG Termes Math. In such cases one must be carefully check whether the size of the glyphs being included fits the size of the basic set of glyphs. Nominally, all fonts (both in the PostScript Type 1 and OTF formats) have the same design size, 10 typographic

Figure 7

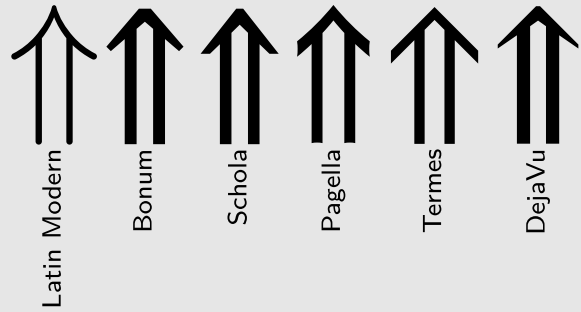
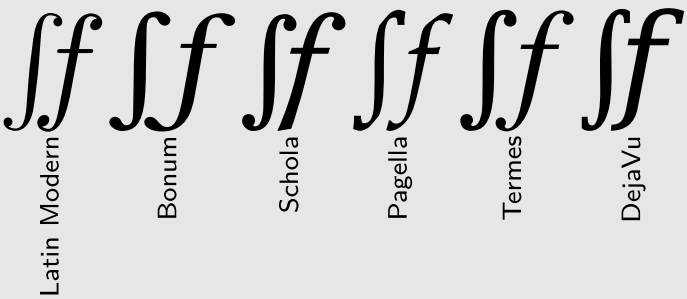


Figure 8



points (recall that 1 point = 1/72 inch; cf. Section 4.4). Working on the TG math fonts, we had to adjust the size of the subsets a few times. For example, we enlarged the borrowed monospaced glyphs to circa 112.5% in TG Schola Math; otherwise the monospaced alphabet looked too small in combination with Schola's brawny glyphs.

The visual harmonizing of the supplementary alphabets with the main font face concerns not only alphanumeric glyphs. Even essentially geometrical shapes should also reflect the characteristic features of the main font, for example, the thickness of stems, the ending of arms, etc. Seemingly trivial glyphs, such as arrows, serve as a convenient example: they have slightly different shapes in each of our math fonts — see Figure 7.

Another example is the shape of integrals. Historically, the integral symbol originates from the letter 'long s'[10] which is nowadays identical with a barless 'f'. Therefore, we did our best to preserve some characteristic features of the letter 'f' in the design of the integral shape — see Figure 8.

[10]The cursive long 's' for denoting an integral operation, i.e., infinite summation, was introduced by Gottfried Wilhelm Leibniz in 1675 in an unpublished paper *Analyseos tetragonisticæ pars secunda* (*Second part of analytical quadrature*).

We are not going to dwell on the visual aspects of the math font design, as the number of details relevant for a font with over 4000 glyphs (and counting) would perhaps become rather overwhelming. Notwithstanding, one aspect needs emphasizing, namely, the problem of sidebearings and kerning for alphanumeric symbols.

It is generally accepted by typographers that math symbols should have larger sidebearings than the respective text glyphs. In particular, TEX math italic glyphs are a little bit broader and have larger sidebearings than the text italic glyphs. It is not obvious, however, how large such sidebearings should be. We have already experimented with a few sizes but further tuning will probably be necessary. Of course, the broadening of sidebearings should not be applied to the basic font, that is, regular upright, because of multiletter names of functions and operators, such as 'sin' or 'max', which are traditionally typeset with regular upright letters.

As regards the problem of kerning, we decided not to include kerns in math fonts, although providing kerns for the upright regular alphabet might be reasonable. Actually, we consider introducing special math kerning (so-called "staircase" kerns a.k.a. "cut-ins"; see Section 7). Thankfully, nobody has complained yet because of the lack of kerning in our math OTF fonts.

6.3. Repertoire issues

At present, a common standard for the repertoire of characters that should be present in an OTF math font does not exist. After several debates (mostly during TEX conferences) and many experiments, we came up with the tentative repertoire scheme presented in Table 2, which can be considered a detailed specification of Table 1.

We would like all TG math fonts (Bonum, DejaVu, Pagella, Schola, and Termes) to share this pattern. For LM Math, however, we adopted another scheme because of legacy concerns. Currently, LM Math contains circa 4800 glyphs while the remaining fonts contain circa 4250 glyphs each. LM Math contains more subscripts, as the original sources already provided them. On the other hand, the TG math fonts contain more size variants of integral symbols. This yields circa 550 glyphs more (net) in LM Math. At the moment, TEX Gyre DejaVu Math contains (in accordance with Table 2) subscript variants for bold upright glyphs, while the remaining TG math fonts need to be complemented with these glyphs. It is a typical instance of the frequently occurring "backtracking" procedures: the final decision was undertaken only after a few fonts had been released.

Table 2

B – basic letters, A – accented letters, G – Greek letters,
D – digits, O – other symbols, P – punctuation

	B	A	G	D	O	P
plain (upright, serif)	$+_{s}$	$+_{s}$	$+^{d}_{s}$	$+_{s}$	$+$	$+_{s}$
italic	$+_{s}$		$+_{s}$			
bold	$+_{s}$		$+^{d}_{s}$	$+_{s}$		
bold italic	$+_{s}$		$+_{s}$			
sans-serif	$+$			$+$		
sans-serif italic	$+$					
sans-serif bold	$+$		$+$	$+$		
sans-serif bold italic	$+$		$+$			
calligraphic	$+$					
bold calligraphic	$+$					
Fraktur	$+$					
bold Fraktur	$+$					
double-struck	$+$			$+$		
monospace	$+$			$+$		

d — digamma excluded from relevant Unicode blocks
s — subscripts are to be added

Concerning subscripts, it should be emphasized that subscript glyphs for the TG math fonts are obtained using the approach applied in the Metafont sources of the Euler family of fonts, that is, by non-uniform rescaling of the respective normal-size glyphs. In a way, such "optical scaling" can be considered undesirable; nevertheless, it proved acceptable for the Euler fonts, thus we decided to apply it also to the TG math fonts.

Some rather quirky characters which received Unicode slots are spaces. Unicode [35] defines quite a few space-related glyphs:

```
 0008 BACKSPACE (<control>)
*0020 SPACE
*00A0 NO-BREAK SPACE
*2002 EN SPACE
*2003 EM SPACE
 1361 ETHIOPIC WORDSPACE
 1680 OGHAM SPACE MARK
*2004 THREE-PER-EM SPACE
*2005 FOUR-PER-EM SPACE
*2006 SIX-PER-EM SPACE
*2007 FIGURE SPACE
*2008 PUNCTUATION SPACE
```

```
*2009  THIN SPACE
*200A  HAIR SPACE
*200B  ZERO WIDTH SPACE
*202F  NARROW NO-BREAK SPACE
*205F  MEDIUM MATHEMATICAL SPACE
 2408  SYMBOL FOR BACKSPACE
 2420  SYMBOL FOR SPACE
 3000  IDEOGRAPHIC SPACE
 303F  IDEOGRAPHIC HALF FILL SPACE
*FEFF  ZERO WIDTH NO-BREAK SPACE
1DA7F  SIGNWRITING LOCATION-WALLPLANE SPACE
1DA80  SIGNWRITING LOCATION-FLOORPLANE SPACE
E0020  TAG SPACE
```

We decided to include a subset of the Unicode-defined spaces (marked with asterisks in the above list), even though their meaning and usage seems vague. The problem of spaces touches a general problem of the relation between the Unicode standard and typography. We discuss this topic in more detail in the next section.

The next two pages show the representative subset (for LM Math and TG Termes Math) of the repertoire we adopted as the GUST e-foundry "private standard"; gray squares denote zero-width characters. As one can see, the repertoires are very similar. One can assume that the difference in the repertoire will be imperceptible in practical applications.

6.4. Unicode: the typographer's friend or enemy?

An important subject, closely related to the contents and repertoire issues, is briefly mentioned in Sections 4.2, 6.1, and 6.3: the problem of the character set defined by the Unicode standard [35], and specified for math fonts in *Unicode Technical Report #25* [37].

The Unicode Consortium claims that "the Unicode standard follows a set of fundamental principles" and gives, among others, the following "principle": *characters, not glyphs* and *semantics*. We are not able to reconcile these principles with the cases discussed in this section.

It should be emphasized that not all glyphs used extensively in typography, in particular, in or out of math formulas, are assigned Unicode numbers. Examples of such glyphs are small caps and old style numerals. Nothing in these two classes of glyphs has received Unicode numbers, although there are codes for much more narrowly used double struck characters or monospaced and sans-serif digits.

Another example of "unicodeless" glyphs are the pieces used for assembling extensible characters (cf. Section 6.1, Figures 5 and 6).

Glyph repertoire of Latin Modern Math

ASCII: ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

Latin-1: ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ´ µ ¶ · ¸ º » ¿ À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

Latin-1 Supplement: Ā ā Ă ă Ą ą Ć ć Ĉ ĉ Ď ď Đ đ Ē ē Ė ė Ę ę Ě ě Ğ ğ Ġ ġ Ī ī Į į İ ı IJ ij Ķ ķ Ĺ Ĺ Ļ ļ Ľ ľ Ł ł Ń ń Ņ ņ Ň ň Ŋ ŋ Ō ō Ő ő Œ œ Ŕ ŕ Ŗ ŗ Ř ř Ś ś Ş ş Š š Ţ ţ Ť ť Ũ ũ Ū ū Ů ů Ű ű Ų ų Ŵ ŵ Ŷ ŷ Ź ź Ż ż Ž ž ſ Ơ ơ Ư ư Ș ș Ț ț ȷ ˆ ˇ ˘ ˙ ˚ ˛ ˜ ˝

Combining Diacritical Marks: ◌̀ ◌́ ◌̂ ◌̃ ◌̄ ◌̆ ◌̇ ◌̈ ◌̉ ◌̊ ◌̋ ◌̌ ◌̑ ◌̓ ◌̔ ◌̛ ◌̣ ◌̤ ◌̥ ◌̦ ◌̧ ◌̨ ◌̩ ◌̪ ◌̫ ◌̬ ◌̭ ◌̮ ◌̯ ◌̰ ◌̱ ◌̲ ◌̳ ◌̴ ◌̵ ◌̶ ◌̷ ◌̸ ◌̡ ◌̢ ◌̺ ◌̻ ◌̼ ◌̽ ◌̾ ◌̿ ◌̀ ◌́ ◌͂ ◌̓ ◌̈́ ◌͆ ◌͇ ◌͈ ◌͉ ◌͊ ◌͋ ◌͌ ◌͍ ◌͎ ◌͐ ◌͑ ◌͒ ◌͓ ◌͔ ◌͕ ◌͖ ◌͗ ◌͘ ◌͙ ◌͚ ◌͛ ◌͜ ◌͝ ◌͞ ◌͟ ◌͠ ◌͡ ◌͢

Greek: Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Φ Χ Ψ Ω α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ ς σ τ υ φ χ ψ ω ϑ ϕ ϖ ϰ ϱ Θ ϵ

Latin Extended Additional: Ạ ạ Ả ả Ấ ấ Ầ ầ Ẩ ẩ Ẫ ẫ Ậ ậ Ắ ắ Ằ ằ Ẳ ẳ Ẵ ẵ Ặ ặ Ẹ ẹ Ẻ ẻ Ẽ ẽ Ế ế Ề ề Ể ể Ễ ễ Ệ ệ Ỉ ỉ Ị ị Ọ ọ Ỏ ỏ Ố ố Ồ ồ Ổ ổ Ỗ ỗ Ộ ộ Ớ ớ Ờ ờ Ở ở Ỡ ỡ Ợ ợ Ụ ụ Ủ ủ Ứ ứ Ừ ừ Ử ử Ữ ữ Ự ự Ỳ ỳ Ỵ ỵ Ỷ ỷ Ỹ ỹ

General Punctuation: ‖ ‗ ‘ ’ ‚ “ ” „ † ‡ • … ‰ ‱ ′ ″ ‴ ‵ ‶ ‷ ‹ › ※ ‽ ⁄ ⁗ ⁽ ⁾

Currency Symbols: ₡ €

Combining Diacritical Marks for Symbols: ◌⃐ ◌⃑ ◌⃒ ◌⃓ ◌⃔ ◌⃕ ◌⃖ ◌⃗ ◌⃘ ◌⃙ ◌⃚ ◌⃛ ◌⃜ ◌⃝ ◌⃞ ◌⃟ ◌⃠ ◌⃡ ◌⃢ ◌⃣ ◌⃤ ◌⃥ ◌⃦ ◌⃧ ◌⃨ ◌⃩ ◌⃪ ◌⃫ ◌⃬ ◌⃭ ◌⃮ ◌⃯ ◌⃰

Letterlike Symbols: ℂ ℃ ℇ ℉ ℋ ℌ ℍ ℎ ℏ ℐ ℑ ℒ ℓ ℕ № ℙ ℘ ℙ ℚ ℛ ℜ ℝ ℞ ™ ℤ Ω ℧ ℨ K Å ℬ ℭ ℮ ℰ ℱ ℳ ℵ ℶ ℷ ℸ ℻ ℼ ℽ ℾ ℿ ⅀ ⅅ ⅆ ⅇ ⅈ ⅉ

Arrows: ← ↑ → ↓ ↔ ↕ ↖ ↗ ↘ ↙ ↚ ↛ ↜ ↝ ↞ ↟ ↠ ↡ ↢ ↣ ↤ ↥ ↦ ↧ ↩ ↪ ↫ ↬ ↭ ↮ ↯ ↰ ↱ ↲ ↳ ↴ ↵ ↶ ↷ ↺ ↻ ↼ ↽ ↾ ↿ ⇀ ⇁ ⇂ ⇃ ⇄ ⇅ ⇆ ⇇ ⇈ ⇉ ⇊ ⇋ ⇌ ⇍ ⇎ ⇏ ⇐ ⇑ ⇒ ⇓ ⇔ ⇕ ⇖ ⇗ ⇘ ⇙ ⇦ ⇧ ⇨ ⇩ ⇳ ⇴ ⇵ ⇶ ⇷ ⇸ ⇹ ⇺ ⇻ ⇼ ⇽ ⇾ ⇿

Mathematical Operators: ∀ ∁ ∂ ∃ ∄ ∅ ∆ ∇ ∈ ∉ ∊ ∋ ∌ ∍ ∎ ∏ ∐ ∑ − ∓ ∔ ∕ ∖ ∗ ∘ ∙ √ ∝ ∞ ∟ ∠ ∡ ∢ ∣ ∤ ∥ ∦ ∧ ∨ ∩ ∪ ∫ ∬ ∭ ⨌ ∮ ∯ ∰ ∱ ∲ ∳ ∴ ∵ ∶ ∷ ∸ ∹ ∺ ∻ ∼ ∽ ∾ ∿ ≀ ≁ ≂ ≃ ≄ ≅ ≆ ≇ ≈ ≉ ≊ ≋ ≌ ≍ ≎ ≏ ≐ ≑ ≒ ≓ ≔ ≕ ≖ ≗ ≘ ≙ ≚ ≛ ≜ ≝ ≞ ≟ ≠ ≡ ≢ ≣ ≤ ≥ ≦ ≧ ≨ ≩ ≪ ≫ ≬ ≭ ≮ ≯ ≰ ≱ ≲ ≳ ≴ ≵ ≶ ≷ ≸ ≹ ≺ ≻ ≼ ≽ ≾ ≿ ⊀ ⊁ ⊂ ⊃ ⊄ ⊅ ⊆ ⊇ ⊈ ⊉ ⊊ ⊋ ⊎ ⊏ ⊐ ⊑ ⊒ ⊓ ⊔ ⊕ ⊖ ⊗ ⊘ ⊙ ⊚ ⊛ ⊜ ⊝ ⊞ ⊟ ⊠ ⊡ ⊢ ⊣ ⊤ ⊥ ⊦ ⊧ ⊨ ⊩ ⊪ ⊫ ⊬ ⊭ ⊮ ⊯ ⊰ ⊱ ⊲ ⊳ ⊴ ⊵ ⊶ ⊷ ⊸ ⊹ ⊺ ⊻ ⊼ ⊽ ⊾ ⊿ ⋀ ⋁ ⋂ ⋃ ⋄ ⋆ ⋇ ⋈ ⋉ ⋊ ⋋ ⋌ ⋍ ⋎ ⋏ ⋐ ⋑ ⋒ ⋓ ⋕ ⋖ ⋗ ⋘ ⋙ ⋚ ⋛ ⋜ ⋝ ⋞ ⋟ ⋠ ⋡ ⋢ ⋣ ⋤ ⋥ ⋦ ⋧ ⋨ ⋩ ⋪ ⋫ ⋬ ⋭ ⋮ ⋯ ⋰ ⋱

Miscellaneous Technical: ∅ ⍼ ⍽ ⌈ ⌉ ⌊ ⌋ ⌐ ⌙ ⌜ ⌝ ⌞ ⌟ ⌠ ⌡ ⌢ ⌣ ⟨ ⟩ ⌀ ⌁ ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋ ⌌ ⌍ ⌎ ⌏ ⌐ ⌑ ⌒ ⌓ ⌔ ⌕ ⌖ ⌗ ⌘ ⌙ ⌚ ⌛ ⌜ ⌝ ⌞ ⌟ ⌠ ⌡ ⌢ ⌣ ⌤ ⌥ ⌦ ⌧ ⌨ 〈 〉 ⌫ ⌬ ⌭ ⌮ ⌯ ⌰ ⌱ ⌲ ⌳ ⌴ ⌵ ⌶ ⌷ ⌸ ⌹ ⌺ ⌻ ⌼ ⌽ ⌾ ⌿ ⍀ ⍁ ⍂ ⍃ ⍄ ⍅ ⍆ ⍇ ⍈ ⍉ ⍊ ⍋ ⍌ ⍍ ⍎ ⍏ ⍐ ⍑ ⍒ ⍓ ⍔ ⍕ ⍖ ⍗ ⍘ ⍙ ⍚ ⍛ ⍜ ⍝ ⍞ ⍟ ⍠ ⍡ ⍢ ⍣ ⍤ ⍥ ⍦ ⍧ ⍨ ⍩ ⍪ ⍫ ⍬ ⍭ ⍮ ⍯ ⍰ ⍱ ⍲ ⍳ ⍴ ⍵ ⍶ ⍷ ⍸ ⍹ ⍺

Box Drawing, Block Elements: ─ │ ┌ ┐ └ ┘ ├ ┤ ┬ ┴ ┼ ▁ ▂ ▃ ▄ ▅ ▆ ▇ █ ▉ ▊ ▋ ▌ ▍ ▎ ▏ ▐ ▒ ▓ ▔ ▕ ▖ ▗ ▘ ▙ ▚ ▛ ▜ ▝ ▞ ▟

Geometric Shapes: ■ □ ▢ ▣ ▤ ▥ ▦ ▧ ▨ ▩ ▪ ▫ ▬ ▭ ▮ ▯ ▰ ▱ ▲ △ ▴ ▵ ▶ ▷ ▸ ▹ ► ▻ ▼ ▽ ▾ ▿ ◀ ◁ ◂ ◃ ◄ ◅ ◆ ◇ ◈ ◉ ○ ◌ ◍ ◎ ● ◐ ◑ ◒ ◓ ◔ ◕ ◖ ◗ ◘ ◙ ◚ ◛ ◜ ◝ ◞ ◟ ◠ ◡ ◢ ◣ ◤ ◥ ◦ ◧ ◨ ◩ ◪ ◫ ◬ ◭ ◮ ◯

Miscellaneous Symbols, Dingbats: ♠ ♡ ♢ ♣ ♤ ♥ ♦ ♧ ♩ ♭ ♮ ♯ ✓ ✠ ❙ ➡

Miscellaneous Mathematical Symbols-A: ⊥ ⟂ ⊤ ⫧ ⫨ ⟛ ⊢ ⟙ ⧫ ⟡ ⟢ ⟣ ⟦ ⟧ ⟨ ⟩ ⟪ ⟫ ⟮ ⟯

Supplemental Arrows-A, B: ⟴ ⟵ ⟶ ⟷ ⟸ ⟹ ⟺ ⟻ ⟼ ⟽ ⟾ ⟿ ⤀ ⤁ ⤂ ⤃ ⤄ ⤅ ⤆ ⤇ ⤈ ⤉ ⤊ ⤋ ⤌ ⤍ ⤎ ⤏ ⤐ ⤑ ⤒ ⤓ ⤔ ⤕ ⤖ ⤗ ⤘ ⤙ ⤚ ⤛ ⤜ ⤝ ⤞ ⤟ ⤠ ⤡ ⤢ ⤣ ⤤ ⤥ ⤦ ⤧ ⤨ ⤩ ⤪ ⤫ ⤬ ⤭ ⤮ ⤯ ⤰ ⤱ ⤲ ⤳ ⤴ ⤵ ⤶ ⤷ ⤸ ⤹ ⤺ ⤻ ⤼ ⤽ ⤾ ⤿ ⥀ ⥁ ⥂ ⥃ ⥄ ⥅ ⥆ ⥇ ⥈ ⥉ ⥊ ⥋ ⥌ ⥍ ⥎ ⥏ ⥐ ⥑ ⥒ ⥓ ⥔ ⥕ ⥖ ⥗ ⥘ ⥙ ⥚ ⥛ ⥜ ⥝ ⥞ ⥟ ⥠ ⥡ ⥢ ⥣ ⥤ ⥥ ⥦ ⥧ ⥨ ⥩ ⥪ ⥫ ⥬ ⥭ ⥮ ⥯ ⥰ ⥱ ⥲ ⥳ ⥴ ⥵ ⥶ ⥷ ⥸ ⥹ ⥺ ⥻ ⥼ ⥽ ⥾ ⥿

Supplemental Mathematical Operators: ⨀ ⨁ ⨂ ⨃ ⨄ ⨅ ⨆ ⨇ ⨈ ⨉ ⨊ ⨋ ⨌ ⨍ ⨎ ⨏ ⨐ ⨑ ⨒ ⨓ ⨔ ⨕ ⨖ ⨗ ⨘ ⨙ ⨚ ⨛ ⨜ ⨝ ⨞ ⨟ ⨠ ⨡ ⨢ ⨣ ⨤ ⨥ ⨦ ⨧ ⨨ ⨩ ⨪ ⨫ ⨬ ⨭ ⨮ ⨯ ⨰ ⨱ ⨲ ⨳ ⨴ ⨵ ⨶ ⨷ ⨸ ⨹ ⨺ ⨻ ⨼ ⨽ ⨾ ⨿ ⩀ ⩁ ⩂ ⩃ ⩄ ⩅ ⩆ ⩇ ⩈ ⩉ ⩊ ⩋ ⩌ ⩍ ⩎ ⩏ ⩐ ⩑ ⩒ ⩓ ⩔ ⩕ ⩖ ⩗ ⩘ ⩙ ⩚ ⩛ ⩜ ⩝ ⩞ ⩟ ⩠ ⩡ ⩢ ⩣ ⩤ ⩥ ⩦ ⩧ ⩨ ⩩ ⩪ ⩫ ⩬ ⩭ ⩮ ⩯ ⩰ ⩱ ⩲ ⩳ ⩴ ⩵ ⩶ ⩷ ⩸ ⩹ ⩺ ⩻ ⩼ ⩽ ⩾ ⩿ ⪀ ⪁ ⪂ ⪃ ⪄ ⪅ ⪆ ⪇ ⪈ ⪉ ⪊ ⪋ ⪌ ⪍ ⪎ ⪏ ⪐ ⪑ ⪒ ⪓ ⪔ ⪕ ⪖ ⪗ ⪘ ⪙ ⪚ ⪛ ⪜ ⪝ ⪞ ⪟ ⪠ ⪡ ⪢ ⪣ ⪤ ⪥ ⪦ ⪧ ⪨ ⪩ ⪪ ⪫ ⪬ ⪭ ⪮ ⪯ ⪰ ⪱ ⪲ ⪳ ⪴ ⪵ ⪶ ⪷ ⪸ ⪹ ⪺ ⪻ ⪼ ⪽ ⪾ ⪿ ⫀ ⫁ ⫂ ⫃ ⫄ ⫅ ⫆ ⫇ ⫈ ⫉ ⫊ ⫋ ⫌ ⫍ ⫎ ⫏ ⫐ ⫑ ⫒ ⫓ ⫔ ⫕ ⫖ ⫗ ⫘ ⫙ ⫚ ⫛ ⫝̸ ⫝ ⫞ ⫟ ⫠ ⫡ ⫢ ⫣ ⫤ ⫥ ⫦ ⫧ ⫨ ⫩ ⫪ ⫫ ⫬ ⫭ ⫮ ⫯ ⫰ ⫱ ⫲ ⫳ ⫴ ⫵ ⫶ ⫷ ⫸ ⫹ ⫺ ⫻ ⫼ ⫽ ⫾ ⫿

Miscellaneous Symbols and Arrows: ⬀ ⬁ ⬂ ⬃ ⬄ ⬅ ⬆ ⬇ ⬈ ⬉ ⬊ ⬋ ⬌ ⬍ ⬎ ⬏ ⬐ ⬑ ⬒ ⬓ ⬔ ⬕ ⬖ ⬗ ⬘ ⬙ ⬚ ⬛ ⬜ ⬝ ⬞ ⬟ ⬠ ⬡ ⬢ ⬣ ⬤ ⬥ ⬦ ⬧ ⬨ ⬩ ⬪ ⬫ ⬬ ⬭ ⬮ ⬯ ⬰ ⬱ ⬲ ⬳ ⬴ ⬵ ⬶ ⬷ ⬸ ⬹ ⬺ ⬻ ⬼ ⬽ ⬾ ⬿ ⭀ ⭁ ⭂ ⭃ ⭄ ⭅ ⭆ ⭇ ⭈ ⭉ ⭊ ⭋ ⭌

CJK Symbols and Punctuation: 〖 〗

Alphabetic Presentation Forms: ﬀ ﬁ ﬂ ﬃ ﬄ

Mathematical Alphanumeric Symbols: 𝐀 𝐁 𝐂 𝐃 𝐄 𝐅 𝐆 𝐇 𝐈 𝐉 𝐊 𝐋 𝐌 𝐍 𝐎 𝐏 𝐐 𝐑 𝐒 𝐓 𝐔 𝐕 𝐖 𝐗 𝐘 𝐙 𝐚 𝐛 𝐜 𝐝 𝐞 𝐟 𝐠 𝐡 𝐢 𝐣 𝐤 𝐥 𝐦 𝐧 𝐨 𝐩 𝐪 𝐫 𝐬 𝐭 𝐮 𝐯 𝐰 𝐱 𝐲 𝐳 𝐴 𝐵 𝐶 𝐷 𝐸 𝐹 𝐺 𝐻 𝐼 𝐽 𝐾 𝐿 𝑀 𝑁 𝑂 𝑃 𝑄 𝑅 𝑆 𝑇 𝑈 𝑉 𝑊 𝑋 𝑌 𝑍 𝑎 𝑏 𝑐 𝑑 𝑒 𝑓 𝑔 𝑖 𝑗 𝑘 𝑙 𝑚 𝑛 𝑜 𝑝 𝑞 𝑟 𝑠 𝑡 𝑢 𝑣 𝑤 𝑥 𝑦 𝑧 𝑨 𝑩 𝑪 𝑫 𝑬 𝑭 𝑮 𝑯 𝑰 𝑱 𝑲 𝑳 𝑴 𝑵 𝑶 𝑷 𝑸 𝑹 𝑺 𝑻 𝑼 𝑽 𝑾 𝑿 𝒀 𝒁 𝒂 𝒃 𝒄 𝒅 𝒆 𝒇 𝒈 𝒉 𝒊 𝒋 𝒌 𝒍 𝒎 𝒏 𝒐 𝒑 𝒒 𝒓 𝒔 𝒕 𝒖 𝒗 𝒘 𝒙 𝒚 𝒛 𝒜 𝒞 𝒟 𝒢 𝒥 𝒦 𝒩 𝒪 𝒫 𝒬 𝒮 𝒯 𝒰 𝒱 𝒲 𝒳 𝒴 𝒵 𝒶 𝒷 𝒸 𝒹 𝒻 𝒽 𝒾 𝒿 𝓀 𝓁 𝓂 𝓃 𝓅 𝓆 𝓇 𝓈 𝓉 𝓊 𝓋 𝓌 𝓍 𝓎 𝓏 𝓐 𝓑 𝓒 𝓓 𝓔 𝓕 𝓖 𝓗 𝓘 𝓙 𝓚 𝓛 𝓜 𝓝 𝓞 𝓟 𝓠 𝓡 𝓢 𝓣 𝓤 𝓥 𝓦 𝓧 𝓨 𝓩 𝓪 𝓫 𝓬 𝓭 𝓮 𝓯 𝓰 𝓱 𝓲 𝓳 𝓴 𝓵 𝓶 𝓷 𝓸 𝓹 𝓺 𝓻 𝓼 𝓽 𝓾 𝓿 𝔀 𝔁 𝔂 𝔃 𝔄 𝔅 𝔇 𝔈 𝔉 𝔊 𝔍 𝔎 𝔏 𝔐 𝔑 𝔒 𝔓 𝔔 𝔖 𝔗 𝔘 𝔙 𝔚 𝔛 𝔜 𝔞 𝔟 𝔠 𝔡 𝔢 𝔣 𝔤 𝔥 𝔦 𝔧 𝔨 𝔩 𝔪 𝔫 𝔬 𝔭 𝔮 𝔯 𝔰 𝔱 𝔲 𝔳 𝔴 𝔵 𝔶 𝔷 𝕬 𝕭 𝕮 𝕯 𝕰 𝕱 𝕲 𝕳 𝕴 𝕵 𝕶 𝕷 𝕸 𝕹 𝕺 𝕻 𝕼 𝕽 𝕾 𝕿 𝖀 𝖁 𝖂 𝖃 𝖄 𝖅 𝖆 𝖇 𝖈 𝖉 𝖊 𝖋 𝖌 𝖍 𝖎 𝖏 𝖐 𝖑 𝖒 𝖓 𝖔 𝖕 𝖖 𝖗 𝖘 𝖙 𝖚 𝖛 𝖜 𝖝 𝖞 𝖟 𝕬 𝕭 𝕮 𝕯 𝕰 𝕱 𝕲 𝕳 𝕴 𝕵 𝕶 𝕷 𝕸 𝕹 𝕺 𝕻 𝕼 𝕽 𝕾 𝕿 𝖀 𝖁 𝖂 𝖃 𝖄 𝖅 𝖆 𝖇 𝖈 𝖉 𝖊 𝖋 𝖌 𝖍 𝖎 𝖏 𝖐 𝖑 𝖒 𝖓 𝖔 𝖕 𝖖 𝖗 𝖘 𝖙 𝖚 𝖛 𝖜 𝖝 𝖞 𝖟 𝟎 𝟏 𝟐 𝟑 𝟒 𝟓 𝟔 𝟕 𝟖 𝟗 𝟘 𝟙 𝟚 𝟛 𝟜 𝟝 𝟞 𝟟 𝟠 𝟡 𝟢 𝟣 𝟤 𝟥 𝟦 𝟧 𝟨 𝟩 𝟪 𝟫 𝟬 𝟭 𝟮 𝟯 𝟰 𝟱 𝟲 𝟳 𝟴 𝟵 𝟶 𝟷 𝟸 𝟹 𝟺 𝟻 𝟼 𝟽 𝟾 𝟿

Glyph repertoire of TEX Gyre Termes Math

ASCII: ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

Latin-1: ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ´ µ ¶ · ¸ º » ¿ À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

Latin-1 Supplement: Ā ā Ă ă Ą ą Ć ć Č č Ď ď Đ đ Ē ē Ė ė Ę ę Ě ě Ğ ğ Ģ ģ Ī ī Į į İ ı IJ ij Ķ ķ Ĺ Ļ ļ Ľ ľ Ł ł Ń ń Ņ ņ Ň ň Ŋ ŋ Ō ō Ő ő Œ œ Ŕ ŗ Ř ř Ś ś Ş ş Š š Ţ ţ Ť ť Ũ ũ Ū ū Ů ů Ű ű Ų ų Ÿ Ź ź Ż ż Ž ž ſ ƒ Ơ ơ Ư ư Ș ș Ț ț ˆ ˇ ˘ ˙ ˚ ˛ ˜ ˝

Combining Diacritical Marks: ◌̀ ◌́ ◌̂ ◌̃ ◌̄ ◌̅ ◌̆ ◌̇ ◌̈ ◌̉ ◌̊ ◌̋ ◌̌ ◌̍ ◌̎ ◌̐ ◌̑ ◌̒ ◌̓ ◌̔ ◌̕ ◌̛ ◌̣ ◌̤ ◌̥ ◌̦ ◌̧ ◌̨ ◌̩ ◌̪ ◌̫ ◌̬ ◌̭ ◌̮ ◌̯ ◌̰ ◌̱ ◌̲ ◌̳ ◌̴ ◌̵ ◌̶ ◌̷ ◌̸ ◌̽ ◌̾ ◌̿ ◌̀ ◌́

Greek: Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Φ Χ Ψ Ω α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ ς σ τ υ φ χ ψ ω ϑ ϕ ϖ ϰ ϱ ϴ ϵ

Latin Extended Additional: Ạ ạ Ả ả Ấ ấ Ầ ầ Ẩ ẩ Ẫ ẫ Ậ ậ Ắ ắ Ằ ằ Ẳ ẳ Ẵ ẵ Ặ ặ Ẹ ẹ Ẻ ẻ Ẽ ẽ Ế ế Ề ề Ể ể Ễ ễ Ệ ệ Ỉ ỉ Ị ị Ọ ọ Ỏ ỏ Ố ố Ồ ồ Ổ ổ Ỗ ỗỘ ộ Ớ ớ Ờ ờ Ở ở Ỡ ỡ Ợ ợ Ụ ụ Ủ ủ Ứ ứ Ừ ừ Ử ử Ữ ữ Ự ự Ỳ ỳ Ỵ ỵ Ỷ ỷ Ỹ ỹ

General Punctuation: ‐ ‑ ‒ – — ― ‖ ‗ ‘ ’ ‚ “ ” „ † ‡ • … ‰ ‱ ′ ″ ‴ ‵ ‶ ‷ ※ ‽ ⁄ ⁇ ⁗ ⁰ ⁽ ⁾ ₀

Currency Symbols: ₡ ₤ ₦ ₩ ₫ € ₱ ₲

Combining Diacritical Marks for Symbols: ◌⃐ ◌⃑ ◌⃒ ◌⃓ ◌⃔ ◌⃕ ◌⃖ ◌⃗ ◌⃘ ◌⃙ ◌⃚ ◌⃛ ◌⃜ ◌⃝ ◌⃞ ◌⃟ ◌⃠ ◌⃡ ◌⃢ ◌⃣ ◌⃤ ◌⃥ ◌⃦ ◌⃧ ◌⃨ ◌⃩ ◌⃪ ◌⃫ ◌⃬ ◌⃭ ◌⃮ ◌⃯ ◌⃰

Letterlike Symbols: ℂ ℃ ℇ ℉ ℊ ℋ ℌ ℍ ℎ ℏ ℐ ℑ ℒ ℓ ℕ № ℗ ℘ ℙ ℚ ℛ ℜ ℝ ℞ ℠ ™ ℤ Ω ℧ ℨ K Å ℬ ℭ ℮ ℯ ℰ ℱ ℳ ℴ ℵ ℶ ℷ ℸ ℼ ℽ ℾ ℿ ⅀ ⅅ ⅆ ⅇ ⅈ ⅉ

Arrows: ← ↑ → ↓ ↔ ↕ ↖ ↗ ↘ ↙ ↚ ↛ ↜ ↝ ↞ ↟ ↠ ↡ ↢ ↣ ↤ ↥ ↦ ↧ ↩ ↪ ↫ ↬ ↭ ↮ ↯ ↰ ↱ ↲ ↳ ↴ ↵ ↶ ↷ ↺ ↻ ↼ ↽ ↾ ↿ ⇀ ⇁ ⇂ ⇃ ⇄ ⇅ ⇆ ⇇ ⇈ ⇉ ⇊ ⇋ ⇌ ⇍ ⇎ ⇏ ⇐ ⇑ ⇒ ⇓ ⇔ ⇕ ⇖ ⇗ ⇘ ⇙ ⇚ ⇛ ⇜ ⇝ ⇦ ⇧ ⇨ ⇩ ⇳ ⇴ ⇵ ⇶ ⇷ ⇸ ⇹ ⇺ ⇻ ⇼ ⇽ ⇾ ⇿

Mathematical Operators: ∀ ∁ ∂ ∃ ∄ ∅ ∆ ∇ ∈ ∉ ∊ ∋ ∌ ∍ ∎ ∏ ∐ ∑ − ∓ ∔ ∕ ∖ ∗ ∘ ∙ √ ∝ ∞ ∟ ∠ ∡ ∢ ∣ ∤ ∥ ∦ ∧ ∨ ∩ ∪ ∫ ∬ ∭ ∮ ∯ ∰ ∱ ∲ ∳ ∴ ∵ ∶ ∷ ∸ ∹ ∺ ∻ ∼ ∽ ∾ ∿ ≀ ≁ ≂ ≃ ≄ ≅ ≆ ≇ ≈ ≉ ≊ ≋ ≌ ≍ ≎ ≏ ≐ ≑ ≒ ≓ ≔ ≕ ≖ ≗ ≘ ≙ ≚ ≛ ≜ ≝ ≞ ≟ ≠ ≡ ≢ ≣ ≤ ≥ ≦ ≧ ≨ ≩ ≪ ≫ ≬ ≭ ≮ ≯ ≰ ≱ ≲ ≳ ≴ ≵ ≶ ≷ ≸ ≹ ≺ ≻ ≼ ≽ ≾ ≿ ⊀ ⊁ ⊂ ⊃ ⊄ ⊅ ⊆ ⊇ ⊈ ⊉ ⊊ ⊋ ⊌ ⊍ ⊎ ⊏ ⊐ ⊑ ⊒ ⊓ ⊔ ⊕ ⊖ ⊗ ⊘ ⊙ ⊚ ⊛ ⊜ ⊝ ⊞ ⊟ ⊠ ⊡ ⊢ ⊣ ⊤ ⊥ ⊦ ⊧ ⊨ ⊩ ⊪ ⊫ ⊬ ⊭ ⊮ ⊯ ⊰ ⊱ ⊲ ⊳ ⊴ ⊵ ⊶ ⊷ ⊸ ⊹ ⊺ ⊻ ⊼ ⊽ ⊾ ⊿ ⋀ ⋁ ⋂ ⋃ ⋄ ⋅ ⋆ ⋇ ⋈ ⋉ ⋊ ⋋ ⋌ ⋍ ⋎ ⋏ ⋐ ⋑ ⋒ ⋓ ⋔ ⋕ ⋖ ⋗ ⋘ ⋙ ⋚ ⋛ ⋜ ⋝ ⋞ ⋟ ⋠ ⋡ ⋢ ⋣ ⋤ ⋥ ⋦ ⋧ ⋨ ⋩ ⋪ ⋫ ⋬ ⋭ ⋮ ⋯ ⋰ ⋱

Miscellaneous Technical: ⌀ ⌅ ⌆ ⌈ ⌉ ⌊ ⌋ ⌐ ⌙ ⌜ ⌝ ⌞ ⌟ ⌠ ⌡ ⌢ ⌣ ⌤ ⌦ ⌧ ⌫ ⌬ ⌭ ⌮ ⌯ ⌰ ⌱ ⌲ ⌳ ⌴ ⌵ ⌶ ⌷ ⌸ ⌹ ⌺ ⌻ ⌼ ⌽ ⌾ ⌿ ⍀ ⍁ ⍂ ⍃ ⍄ ⍅ ⍆ ⍇ ⍈ ⍉ ⍊ ⍋ ⍌ ⍍ ⍎ ⍏ ⍐ ⍑ ⍒ ⍓ ⍔ ⍕ ⍖ ⍗ ⍘ ⍙ ⍚ ⍛ ⍜ ⍝ ⍞ ⍟ ⍠ ⍡ ⍢ ⍣ ⍤ ⍥ ⍦ ⍧ ⍨ ⍩ ⍪ ⍫ ⍬ ⍭ ⍮ ⍯ ⍰ ⍱ ⍲ ⍳ ⍴ ⍵ ⍶ ⍷ ⍸ ⍹ ⍺

Box Drawing, Block Elements: ─ │ ┌ ┐ └ ┘ ├ ┤ ┬ ┴ ┼ ▁ ▇ ▒ ▓ ▩

Geometric Shapes: ■ □ ▪ ▫ ▬ ▭ ▲ △ ▶ ▷ ▼ ▽ ◀ ◁ ◆ ◇ ○ ● ◦ ◯

Miscellaneous Symbols, Dingbats: ♠ ♡ ♢ ♣ ♤ ♥ ♦ ♧ ♪ ♭ ♮ ♯ ✓ ✠ ➞

Miscellaneous Mathematical Symbols-A: ⟀ ⟂ ⟒ ⟛ ⟡ ⟢ ⟣ ⟤ ⟥ ⟦ ⟧ ⟨ ⟩ ⟪ ⟫ ⟮ ⟯

Supplemental Arrows-A, B: ⟰ ⟱ ⟲ ⟳ ⟴ ⟵ ⟶ ⟷ ⟸ ⟹ ⟺ ⟻ ⟼ ⟽ ⟾ ⟿ ⤀ ⤁ ⤂ ⤃ ⤄ ⤅ ⤆ ⤇ ⤈ ⤉ ⤊ ⤋ ⤌ ⤍ ⤎ ⤏ ⤐ ⤑ ⤒ ⤓ ⤔ ⤕ ⤖ ⤗ ⤘ ⤙ ⤚ ⤛ ⤜ ⤝ ⤞ ⤟ ⤠ ⤡ ⤢ ⤣ ⤤ ⤥ ⤦ ⤧ ⤨ ⤩ ⤪ ⤫ ⤬ ⤭ ⤮ ⤯ ⤰ ⤱ ⤲ ⤳ ⤴ ⤵ ⤶ ⤷ ⤸ ⤹ ⤺ ⤻ ⤼ ⤽ ⤾ ⤿ ⥀ ⥁ ⥂ ⥃ ⥄ ⥅ ⥆ ⥇ ⥈ ⥉ ⥊ ⥋ ⥌ ⥍ ⥎ ⥏ ⥐ ⥑ ⥒ ⥓ ⥔ ⥕ ⥖ ⥗ ⥘ ⥙ ⥚ ⥛ ⥜ ⥝ ⥞ ⥟ ⥠ ⥡ ⥢ ⥣ ⥤ ⥥ ⥦ ⥧ ⥨ ⥩ ⥪ ⥫ ⥬ ⥭ ⥮ ⥯ ⥰ ⥱ ⥲ ⥳ ⥴ ⥵ ⥶ ⥷ ⥸ ⥹ ⥺ ⥻ ⥼ ⥽ ⥾ ⥿

Supplemental Mathematical Operators: ⨀ ⨁ ⨂ ⨃ ⨄ ⨅ ⨆ ⨇ ⨈ ⨉ ⨊ ⨋ ⨌ ⨍ ⨎ ⨏ ⨐ ⨑ ⨒ ⨓ ⨔ ⨕ ⨖ ⨗ ⨘ ⨙ ⨚ ⨛ ⨜ ⨝ ⨞ ⨟ ⨠ ⨡ ⨢ ⨣ ⨤ ⨥ ⨦ ⨧ ⨨ ⨩ ⨪ ⨫ ⨬ ⨭ ⨮ ⨯ ⨰ ⨱ ⨲ ⨳ ⨴ ⨵ ⨶ ⨷ ⨸ ⨹ ⨺ ⨻ ⨼ ⨽ ⨾ ⨿ ⩀ ⩁ ⩂ ⩃ ⩄ ⩅ ⩆ ⩇ ⩈ ⩉ ⩊ ⩋ ⩌ ⩍ ⩎ ⩏ ⩐ ⩑ ⩒ ⩓ ⩔ ⩕ ⩖ ⩗ ⩘ ⩙ ⩚ ⩛ ⩜ ⩝ ⩞ ⩟ ⩠ ⩡ ⩢ ⩣ ⩤ ⩥ ⩦ ⩧ ⩨ ⩩ ⩪ ⩫ ⩬ ⩭ ⩮ ⩯ ⩰ ⩱ ⩲ ⩳ ⩴ ⩵ ⩶ ⩷ ⩸ ⩹ ⩺ ⩻ ⩼ ⩽ ⩾ ⩿ ⪀ ⪁ ⪂ ⪃ ⪄ ⪅ ⪆ ⪇ ⪈ ⪉ ⪊ ⪋ ⪌ ⪍ ⪎ ⪏ ⪐ ⪑ ⪒ ⪓ ⪔ ⪕ ⪖ ⪗ ⪘ ⪙ ⪚ ⪛ ⪜ ⪝ ⪞ ⪟ ⪠ ⪡ ⪢ ⪣ ⪤ ⪥ ⪦ ⪧ ⪨ ⪩ ⪪ ⪫ ⪬ ⪭ ⪮ ⪯ ⪰ ⪱ ⪲ ⪳ ⪴ ⪵ ⪶ ⪷ ⪸ ⪹ ⪺ ⪻ ⪼ ⪽ ⪾ ⪿ ⫀ ⫁ ⫂ ⫃ ⫄ ⫅ ⫆ ⫇ ⫈ ⫉ ⫊ ⫋ ⫌ ⫍ ⫎ ⫏ ⫐ ⫑ ⫒ ⫓ ⫔ ⫕ ⫖ ⫗ ⫘ ⫙ ⫚ ⫛ ⫝̸ ⫝ ⫞ ⫟ ⫠ ⫡ ⫢ ⫣ ⫤ ⫥ ⫦ ⫧ ⫨ ⫩ ⫪ ⫫ ⫬ ⫭ ⫮ ⫯ ⫰ ⫱ ⫲ ⫳ ⫴ ⫵ ⫶ ⫷ ⫸ ⫹ ⫺ ⫻ ⫼ ⫽ ⫾ ⫿

Miscellaneous Symbols and Arrows: ⬀ ⬁ ⬂ ⬃ ⬄ ⬅ ⬆ ⬇ ⬈ ⬉ ⬊ ⬋ ⬌ ⬍ ⬎ ⬏ ⬐ ⬑ ⬒ ⬓ ⬔ ⬕ ⬖ ⬗ ⬘ ⬙ ⬚ ⬛ ⬜ ⬝ ⬞ ⬟ ⬠ ⬡ ⬢ ⬣ ⬤ ⬥ ⬦ ⬧ ⬨ ⬩ ⬪ ⬫ ⬬ ⬭ ⬮ ⬯ ⬰ ⬱ ⬲ ⬳ ⬴ ⬵ ⬶ ⬷ ⬸ ⬹ ⬺ ⬻ ⬼ ⬽ ⬾ ⬿ ⭀ ⭁ ⭂ ⭃ ⭄ ⭅ ⭆ ⭇ ⭈ ⭉ ⭊ ⭋ ⭌ ⭍ ⭎ ⭏ ⭐ ⭑ ⭒ ⭓ ⭔ ⭕

CJK Symbols and Punctuation: 〖 〗

Alphabetic Presentation Forms: ff fi fl ffi ffl

Mathematical Alphanumeric Symbols: **A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z** *A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g i j k l m n o p q r s t u v w x y z* ***A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z*** *𝒜 ℬ 𝒞 𝒟 𝒢 𝒥 𝒦 𝒩 𝒪 𝒫 𝒬 𝒮 𝒯 𝒰 𝒱 𝒲 𝒳 𝒴 𝒵 𝒶 𝒷 𝒸 𝒹 𝒻 𝒽 𝒾 𝒿 𝓀 𝓁 𝓂 𝓃 𝓅 𝓆 𝓇 𝓈 𝓉 𝓊 𝓋 𝓌 𝓍 𝓎 𝓏* ***𝒜 ℬ 𝒞 𝒟 ℰ ℱ 𝒢 𝒽 ℐ 𝒥 𝒦 ℒ 𝓂 𝒩 𝒪 𝒫 𝒬 ℛ 𝒮 𝒯 𝒰 𝒱 𝒲 𝒳 𝒴 𝒵 𝒶 𝒷 𝒸 𝒹 ℯ 𝒻 𝑔 𝒽 𝒾 𝒿 𝓀 𝓁 𝓂 𝓃 𝑜 𝓅 𝓆 𝓇 𝓈 𝓉 𝓊 𝓋 𝓌 𝓍 𝓎 𝓏*** 𝔄 𝔅 ℭ 𝔇 𝔈 𝔉 𝔊 ℌ ℑ 𝔎 𝔏 𝔐 𝔑 𝔒 𝔓 𝔔 ℜ 𝔖 𝔗 𝔘 𝔙 𝔚 𝔛 𝔜 ℨ 𝔞 𝔟 𝔠 𝔡 𝔢 𝔣 𝔤 𝔥 𝔦 𝔧 𝔨 𝔩 𝔪 𝔫 𝔬 𝔭 𝔮 𝔯 𝔰 𝔱 𝔲 𝔳 𝔴 𝔵 𝔶 𝔷 𝔸 𝔹 ℂ 𝔻 𝔼 𝔽 𝔾 ℍ 𝕀 𝕁 𝕂 𝕃 𝕄 ℕ 𝕆 ℙ ℚ ℝ 𝕊 𝕋 𝕌 𝕍 𝕎 𝕏 𝕐 ℤ 𝕒 𝕓 𝕔 𝕕 𝕖 𝕗 𝕘 𝕙 𝕚 𝕛 𝕜 𝕝 𝕞 𝕟 𝕠 𝕡 𝕢 𝕣 𝕤 𝕥 𝕦 𝕧 𝕨 𝕩 𝕪 𝕫 𝔸 𝔹 𝔻 𝔼 𝔽 𝔾 𝕀 𝕁 𝕂 𝕃 𝕄 𝕆 𝕊 𝕋 𝕌 𝕍 𝕎 𝕏 𝕐 𝕒 𝕓 𝕔 𝕕 𝕖 𝕗 𝕘 𝕙 𝕚 𝕛 𝕜 𝕝 𝕞 𝕟 𝕠 𝕡 𝕢 𝕣 𝕤 𝕥 𝕦 𝕧 𝕨 𝕩 𝕪 𝕫 𝔸 𝔹 𝔻 𝔼 𝔽 𝔾 𝕀 𝕁 𝕂 𝕃 𝕄 𝕆 𝕊 𝕋 𝕌 𝕍 𝕎 𝕏 𝕐 𝖆 𝖇 𝖈 𝖉 𝖊 𝖋 𝖌 𝖍 𝖎 𝖏 𝖐 𝖑 𝖒 𝖓 𝖔 𝖕 𝖖 𝖗 𝖘 𝖙 𝖚 𝖛 𝖜 𝖝 𝖞 𝖟 𝕬 𝕭 𝕮 𝕯 𝕰 𝕱 𝕲 𝕳 𝕴 𝕵 𝕶 𝕷 𝕸 𝕹 𝕺 𝕻 𝕼 𝕽 𝕾 𝕿 𝖀 𝖁 𝖂 𝖃 𝖄 𝖅 𝖠 𝖡 𝖢 𝖣 𝖤 𝖥 𝖦 𝖧 𝖨 𝖩 𝖪 𝖫 𝖬 𝖭 𝖮 𝖯 𝖰 𝖱 𝖲 𝖳 𝖴 𝖵 𝖶 𝖷 𝖸 𝖹 𝟎 𝟏 𝟐 𝟑 𝟒 𝟓 𝟔 𝟕 𝟖 𝟗 **𝟎 𝟏 𝟐 𝟑 𝟒 𝟓 𝟔 𝟕 𝟖 𝟗** 𝟘 𝟙 𝟚 𝟛 𝟜 𝟝 𝟞 𝟟 𝟠 𝟡 **𝟘 𝟙 𝟚 𝟛 𝟜 𝟝 𝟞 𝟟 𝟠 𝟡** 𝟢 𝟣 𝟤 𝟥 𝟦 𝟧 𝟨 𝟩 𝟪 𝟫

It is not so bad if whole blocks of characters are included or excluded. The worse situation is an inconsistency of including only some glyphs. The above-mentioned double struck glyphs are a good example of such a situation. Here is the group of double struck glyphs assigned Unicode slots, without apparent rhyme or reason:

```
2102 DOUBLE-STRUCK CAPITAL C
210D DOUBLE-STRUCK CAPITAL H
2115 DOUBLE-STRUCK CAPITAL N
2119 DOUBLE-STRUCK CAPITAL P
211A DOUBLE-STRUCK CAPITAL Q
211D DOUBLE-STRUCK CAPITAL R
2124 DOUBLE-STRUCK CAPITAL Z
213C DOUBLE-STRUCK SMALL PI
213D DOUBLE-STRUCK SMALL GAMMA
213E DOUBLE-STRUCK CAPITAL GAMMA
213F DOUBLE-STRUCK CAPITAL PI
2140 DOUBLE-STRUCK N-ARY SUMMATION
2145 DOUBLE-STRUCK ITALIC CAPITAL D
2146 DOUBLE-STRUCK ITALIC SMALL D
2147 DOUBLE-STRUCK ITALIC SMALL E
2148 DOUBLE-STRUCK ITALIC SMALL I
2149 DOUBLE-STRUCK ITALIC SMALL J
```

The remaining letters of the alphabet (upper and lower case) and digits received mathematical codes (1D538 – 1D56B), with gaps at the glyphs above.

Another notable example are sub- and superscripts, theoretically not needed in math fonts, because the sub- and superscript characters ("unicodeless") are accessed there by the OTF feature mechanism (the `ssty` feature, cf. Section 6.1). In practice, for legacy reasons, sub- and superscript glyphs are expected to be present in a text font, and, consequently, in the basic (plain) charset of a math font too — see Table 1. The Unicode standard [35] enumerates the following Latin letters in this context:

```
1D62 LATIN SUBSCRIPT SMALL LETTER I
1D63 LATIN SUBSCRIPT SMALL LETTER R
1D64 LATIN SUBSCRIPT SMALL LETTER U
1D65 LATIN SUBSCRIPT SMALL LETTER V
2090 LATIN SUBSCRIPT SMALL LETTER A
2091 LATIN SUBSCRIPT SMALL LETTER E
2092 LATIN SUBSCRIPT SMALL LETTER O
2093 LATIN SUBSCRIPT SMALL LETTER X
2094 LATIN SUBSCRIPT SMALL LETTER SCHWA
2095 LATIN SUBSCRIPT SMALL LETTER H
```

```
2096 LATIN SUBSCRIPT SMALL LETTER K
2097 LATIN SUBSCRIPT SMALL LETTER L
2098 LATIN SUBSCRIPT SMALL LETTER M
2099 LATIN SUBSCRIPT SMALL LETTER N
209A LATIN SUBSCRIPT SMALL LETTER P
209B LATIN SUBSCRIPT SMALL LETTER S
209C LATIN SUBSCRIPT SMALL LETTER T
2C7C LATIN SUBSCRIPT SMALL LETTER J

2071 SUPERSCRIPT LATIN SMALL LETTER I
207F SUPERSCRIPT LATIN SMALL LETTER N
```

Besides the somewhat surprising presence of the 'schwa' character and the striking asymmetry between the number of sub- and superscripts, most questionable here is the incompleteness of the Latin alphabet. So far, we have not included these glyphs in neither text nor math fonts.

Another example concerns math italic symbols. The Unicode standard reads:

```
1D44E MATHEMATICAL ITALIC SMALL A
1D44F MATHEMATICAL ITALIC SMALL B
1D450 MATHEMATICAL ITALIC SMALL C
1D451 MATHEMATICAL ITALIC SMALL D
1D452 MATHEMATICAL ITALIC SMALL E
1D453 MATHEMATICAL ITALIC SMALL F
1D454 MATHEMATICAL ITALIC SMALL G
                                 ⇐?
1D456 MATHEMATICAL ITALIC SMALL I
1D457 MATHEMATICAL ITALIC SMALL J
1D458 MATHEMATICAL ITALIC SMALL K
1D459 MATHEMATICAL ITALIC SMALL L
1D45A MATHEMATICAL ITALIC SMALL M
1D45B MATHEMATICAL ITALIC SMALL N
1D45C MATHEMATICAL ITALIC SMALL O
1D45D MATHEMATICAL ITALIC SMALL P
1D45E MATHEMATICAL ITALIC SMALL Q
1D45F MATHEMATICAL ITALIC SMALL R
1D460 MATHEMATICAL ITALIC SMALL S
1D461 MATHEMATICAL ITALIC SMALL T
1D462 MATHEMATICAL ITALIC SMALL U
1D463 MATHEMATICAL ITALIC SMALL V
1D464 MATHEMATICAL ITALIC SMALL W
1D465 MATHEMATICAL ITALIC SMALL X
1D466 MATHEMATICAL ITALIC SMALL Y
1D467 MATHEMATICAL ITALIC SMALL Z
```

Math italic ‘h’ is apparently missing. In fact, the Unicode Consortium decided to leave the slot `1D455` unused forever and “inflate” the meaning of the slot formerly assigned to the Planck constant:

```
210E PLANCK CONSTANT
     = height, specific enthalpy, ...
     * simply a mathematical italic h;
       this character's name results
       from legacy usage
```

More on Unicode’s ambiguities, inconsistencies, riddles, curiosities, etc. concerning typography applications, especially math typesetting, can be found in Piotr Strzelczyk’s ruminations [23].

From the typographer’s point of view, the enumeration of all signs used in the world does not seem to be a good idea. Therefore, it should be no surprise that it turned out to be impossible to create a math OTF font that has an internally logical and coherent structure and, at the same time, is practically useful. Conforming rigorously to the mentioned specifications does not help too much — it would lead, in our opinion, to fonts containing mostly seldom used glyphs. Needless to say, research examining which characters from the Unicode repertoire are actually used in practice, would be welcome. Lacking such empirical data, we adopted Cambria Math as our “reference point”. Cambria Math contains circa 6500 glyphs; we decided to reduce this number to at most 4500 for the TEX Gyre series of math fonts. We can reluctantly consider proposals for suitable extensions, if truly needed.

6.5. Compatibility issues

We already explained why 100-percent compatibility of the LM text fonts with the parent Computer Modern fonts is unfeasible; thereby, the same applies, even to a greater extent, to the LM Math font (cf. the case of horizontal extensible braces in Section 6.1).

Recall that we used an alternative calligraphic alphabet in LM Math (Section 6.1). This incompatibility can be relatively easily patched, by including two calligraphic scripts in the font and using OTF ‘stylistic sets’ features, implemented as the OTF features `ss01` – `ss20` (see [33]).

There is, however, a flaw shared by both Computer Modern and Euler calligraphic alphabets and thus inherited by the Latin Modern Math: the lack of a corresponding lower case alphabet. The Unicode specification assigns slots to lower case mathematical calligraphic letters, implying that they are expected to be present in math fonts.

We could not find a calligraphic font with a proper license optically matching Euler or Computer Modern upper case calligraphic letters, and we gave up in advance the idea of designing the respective matching lower case letters ourselves.

Figure 9

Instead, we began to consider the inclusion of yet another stylistic set, a "home-made" calligraphic font inspired by (but not based on) the original Computer Modern calligraphic script. Although we would never dare to make a text calligraphic font without close cooperation by a professional type designer, we took a chance and attempted to prepare a symbol calligraphic font for TG DejaVu Math.

There was an additional reason for doing our own calligraphic script. Anticipating future work (see Section 7 below), we looked for a calligraphic script matching a sans-serif font. Having not found any, we decided to prepare our own. Because our calligraphic alphabet for TG DejaVu Math was, of course, defined by a parametric MetaType 1 program, it was possible to prepare a sans-serif (linear) variant of the calligraphic script with reasonable effort. A tentative, experimental linear calligraphic alphabet is shown in Figure 9: the calligraphic script used in TG DejaVu Math (top) and its linear variant to be used in a sans-serif math font (bottom).

Fortunately, there is no issue of compatibility regarding the TG math fonts, as there are no predecessors. The only question is the similarity of repertoire between the LM and TG math fonts. As explained in Section 6.3, the repertoires are bound to differ slightly, for the usual legacy reasons.

For the same reasons, some technical details of the font structure also cannot be implemented similarly. Worthy of mentioning in this context is a peculiar difference between the LM and TG math fonts related to integrals; namely, the TG math fonts contain extensible integrals, which are definable within the OTF math format — but we are not aware of any typesetting engine that can take advantage of this possibility.

7. Plans for the future

7.1. Testing and maintenance

Tasks that are important today and will be forever important in the future are maintenance and testing. There is, of course, neither a single tool for testing nor a unique maintenance procedure. Each case demands a specific approach.

It is Piotr Pianowski who is responsible for testing fonts and preparing adequate tools. Tests refer to both the appearance of the fonts and their internal structure. In particular, the intermediate PostScript Type 1 code needs checking. The following example shows a case where the inspection of the code revealed an error in the `parenright.ex` procedure (describing an extender of the extensible right parenthesis, which should be just a rectangle), probably due to the wrong rounding procedure: the number '1' means that the line drawn by the command `rlineto` is not strictly vertical. The left column contains the correct code for the corresponding component of the extensible left parenthesis:

```
/parenleft.ex {     /parenright.ex {
  143 609 hsbw        338 609 hsbw
  127 418 rmoveto     128 418 rmoveto
  -127 hlineto        -128 hlineto
  -418 vlineto        1 -418 rlineto
  127 hlineto         126 hlineto
  closepath           closepath
  endchar             endchar
  } ND                } ND
```

It is next to impossible to perceive such a tiny deformity on a printout of glyph shapes, which does not mean that the bug should not be fixed.

The next example, Figure 10, shows one of the tests we use for checking a vexing problem regarding Greek letter names. Some Greek letters have a shape variant, and there is an unfortunate discrepancy between TEXies and the rest of the world (the rightmost two columns marked with asterisks) as to which glyphs are considered "normal" and which "variant".

Almost every time we deal with the Greek alphabet, a mistake in variant names tries to creep in. Without tests of this kind we would be lost.

The last example concerning maintenance and testing issues, Figure 11, shows a typical test of the structure of extensible characters, "embellished" horizontal arrows in this case: 'vh 2' means that there are 2 size variants, 'ch 3' and 'ch 5' — that there are 3 and 5 horizontal components of a given glyph, respectively. Tests of this kind are essential for maintaining uniformity across a font collection as well as inside a single font.

As a result of remarks to date from the users of our fonts, we gathered a list of recommended fixes and improvements — some trivial, like reports on malformed glyphs or wrongly assigned Unicode slots, some fairly difficult, like suggestions to

texgyredejavu-math.otf

	normal	var	normal*	var*
\mstd	$\epsilon\ \phi\ \rho\ \theta$	$\varepsilon\ \varphi\ \varrho\ \vartheta$	$\varepsilon\ \varphi\ \rho\ \theta$	$\epsilon\ \phi\ \varrho\ \vartheta$
\mrm	$\epsilon\ \varphi\ \rho\ \theta$	$\varepsilon\ \phi\ \varrho\ \vartheta$	$\varepsilon\ \phi\ \rho\ \theta$	$\epsilon\ \varphi\ \varrho\ \vartheta$
\mit	$\epsilon\ \phi\ \rho\ \theta$	$\varepsilon\ \varphi\ \varrho\ \vartheta$	$\varepsilon\ \varphi\ \rho\ \theta$	$\epsilon\ \phi\ \varrho\ \vartheta$
\mbf	$\boldsymbol{\epsilon\ \varphi\ \rho\ \theta}$	$\boldsymbol{\varepsilon\ \phi\ \varrho\ \vartheta}$	$\boldsymbol{\varepsilon\ \phi\ \rho\ \theta}$	$\boldsymbol{\epsilon\ \varphi\ \varrho\ \vartheta}$
\mbi	$\boldsymbol{\epsilon\ \phi\ \rho\ \theta}$	$\boldsymbol{\varepsilon\ \varphi\ \varrho\ \vartheta}$	$\boldsymbol{\varepsilon\ \varphi\ \rho\ \theta}$	$\boldsymbol{\epsilon\ \phi\ \varrho\ \vartheta}$
\mbfss	$\boldsymbol{\epsilon\ \phi\ \rho\ \theta}$	$\boldsymbol{\varepsilon\ \varphi\ \varrho\ \vartheta}$	$\boldsymbol{\varepsilon\ \varphi\ \rho\ \theta}$	$\boldsymbol{\epsilon\ \phi\ \varrho\ \vartheta}$
\mitss	$\epsilon\ \phi\ \rho\ \theta$	$\varepsilon\ \varphi\ \varrho\ \vartheta$	$\varepsilon\ \varphi\ \rho\ \theta$	$\epsilon\ \phi\ \varrho\ \vartheta$
\mbiss	$\boldsymbol{\epsilon\ \phi\ \rho\ \theta}$	$\boldsymbol{\varepsilon\ \varphi\ \varrho\ \vartheta}$	$\boldsymbol{\varepsilon\ \varphi\ \rho\ \theta}$	$\boldsymbol{\epsilon\ \phi\ \varrho\ \vartheta}$

Figure 10

texgyredejavu-math.otf

uni219A vh 2 ch 5

uni21A9 vh 2 ch 3

uni21AB vh 2 ch 3

Figure 11

implement anchors or math staircase kerns.

The latter two potential improvements are still pending, mainly because of vague specification and the question of practical application. Being unsure whether all relevant typesetting programs would be able to handle such improvements, we preferred to linger till the engines would become stable. It seems that the time is ripe to attempt these extensions, especially as the staircase kerns were ultimately implemented in XeTeX in the middle of 2014.

We are also not sure which OTF features should be present in math fonts. At the moment, only math-oriented OTF features are implemented. Perhaps we should consider the inclusion of text-oriented OTF features too, like the mentioned

onum, lnum, pnum, and tnum for switching between numeral variants, or stylistic sets for switching between calligraphic alphabets.

Also as suggested by users, we consider excerpting a number of glyphs, mostly geometrical shapes and arrows but also selected math operators and relational symbols, etc. (but excluding extensible and subscript characters), from math fonts and transferring them into the respective text fonts. Such glyphs, though prepared for math-oriented applications, can also be fruitfully used in conventional fonts, that is, those not equipped with the MATH table, for the typesetting of technical documents. This, obviously, requires a proper specification and careful selection of the glyphs in question. This is, in fact, one of the planned stages of work on new fonts.

Of course, MetaType 1 itself also requires maintenance: enhancing, modifying, fixing, etc. For example, we had to extend the otherwise stable set of MetaPost macros used in MetaType 1 in order to handle the math extension of the OTF specification.

Commencing our works for the GUST e-foundry, we underestimated the inexorable Hofstadter's Law (see footnote 7 on page 21) which apparently applies not only to time resources. We believed that MetaType 1 could be kept simple: just MetaPost, a few trivial Gawk scripts and a stand-alone, stable converter to PostScript Type 1 fonts, T1utils, and that's all. In accordance with Hofstadter's Law, the task has unavoidably turned out to be much more complex than we expected and the implementation — too heterogeneous.

In order to remedy this, we intend to unify MetaType 1 by eliminating Gawk, Perl, T1utils, and, as we mentioned in Section 3.1, AFDKO. We aim at employing two basic "subengines", namely, MetaPost (for generating outlines) and Python (for arranging the MetaPost output) plus one external, possibly replaceable, "subengine", for now, the FontForge Python library (for generating OTF fonts and the theoretically obsolescent but still widely used PostScript Type 1 fonts).

7.2. More GUST e-foundry fonts

In the nearest future, we would like to elaborate and release three experimental math fonts, namely: bold math, sans-serif math, and monospaced math. These fonts fit neither the Microsoft nor Unicode specifications ([29] and [37], respectively). Therefore, we have to start by working out an altered specification, adjusted to our purposes, for these non-standard cases. For example, it is not at all clear what to do with the sans-serif alphabet in a sans-serif math font: should it be omitted, left intact, or modified somehow (how?).

Bold math fonts can be used in titles containing math formulas, as shown in Figure 12 (a tentative version of TG Termes Bold Math is used).

Nowadays, many documents are being typeset in sans-serif, even schoolbooks. Computer presentations may serve as another example. For such applications

Figure 12

1. Equivalence of the integral $\left(\oiint_{\partial S} \mathbf{E} \cdot d\mathbf{S} = 0\right)$ and differential $(\nabla \cdot \mathbf{E} = 0)$ formulas

In this section, we shall prove that the integral and differential forms of Maxwell's equation known as "Gauss's law for magnetism", i.e.:

$$\oiint_{\partial S} \mathbf{E} \cdot d\mathbf{S} = 0 \tag{1}$$

and

$$\nabla \cdot \mathbf{E} = 0 \tag{2}$$

are equivalent.

Figure 13

Various notations for continued fractions

The integers a_0, a_1 etc., are called *coefficients* or *terms* of the continued fraction. One can abbreviate the continued fraction

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3}}}$$

in the notation of Carl Friedrich Gauss as

$$x = a_0 + \mathop{\mathrm{K}}_{i=1}^{3} \frac{1}{a_i}$$

or in the notation of Alfred Pringsheim as

$$x = a_0 + \frac{1}{|a_1|} + \frac{1}{|a_2|} + \frac{1}{|a_3|}.$$

source: http://en.wikipedia.org/wiki/Continued_fraction

sans-serif math seems plausible. An example of such an application is shown in Figure 13 (a tentative version of TG DejaVu Sans Math is used).

TEX users are accustomed to the use of control sequences for math-oriented glyphs such as `\infty`, `\sum` or `\pm`. Some of these glyphs can be accessed (typed in and displayed) directly in text editors, provided the font used by the editor contains the relevant glyphs. The lion's share of nominally math-oriented glyphs can also be prepared as monospaced glyphs, usable in text editors, provided they are assigned Unicode numbers. One can expect that it will improve the legibility

```
\section{Алгоритм де Кастельє}

Заданий многочлен Бернштейна $B$ ступеня $n$
з опорними точками $β₀,…,βₙ$
$$
  B(t) = ∑ⁿᵢ₌₀ βᵢb_{i,n}(t),
$$
де $b$ — базис многочлена Бернштейна, многочлен
в точці $t₀$ може бути визначений за допомогою
рекурентного співвідношення:
$$\eqalign{
  βᵢ⁽⁰⁾& ≔ βᵢ \mbox{, } i=0,…,n\cr
  βᵢ^{(j)}& ≔ βᵢ^{(j-1)}(1-t₀) + βᵢ₊₁^{(j-1)}t₀
     \mbox{, } i=0,…,n-j \mbox{, } j=1,…,n\cr
}$$
Тоді визначення $B$ в точці $t₀$ може бути
визначено в $n$ кроків алгоритму.
Результат $B(t₀)$ дано за:
$$
  B(t₀)=β₀⁽ⁿ⁾.
$$
Також, крива Безьє $B$ може бути розділена
в точці $t₀$ на дві криві з відповідними
опорними точками:
$$
  β₀⁽⁰⁾,β₀⁽¹⁾,…,β₀⁽ⁿ⁾
  β₀⁽ⁿ⁾,β₁⁽ⁿ⁻¹⁾,…,βₙ⁽⁰⁾
$$
```

Figure 14

source: https://uk.wikipedia.org/wiki/Алгоритм_де_Кастельє

of sources and, thereby, the efficiency of preparation of such documents — see Figure 14 (a tentative version of TG DejaVu Mono Math is used).

Note that treating subscripts incoherently by the Unicode Standard (Section 6.4) may prove to be a significant impediment in the latter case, that is, for fonts without full subscript support. Perhaps the defining of the complete set of subscripts and superscripts (partially "unicodeless") and accessing them via stylistic set features (`ss01` – `ss20`) can be a solution, provided a given text editor handles the relevant OTF features.

7.3. Legal issues

The problem of copyrights and licenses has clung to us like a leech from the very beginning and still persists. We are not copyright experts and we do not want to be. Being sick of trouble with releasing our work due to discussions about legal

aspects of distributing free fonts, we coined a pun *lice-sense*. Just one example: the release of TG DejaVu Math was delayed by about a year because of doubts raised concerning legal matters.

Having said this, we should emphasize that it does not mean that there has been no activity from the side of the GUST e-foundry regarding legal issues. On the contrary, our magnificent liaison officer, Jerzy Ludwichowski, has made Herculean efforts in order to provide appropriate licenses for GUST fonts. In particular, he prepared a proposal of a license for the URW fonts that would suit GUST e-foundry needs, managed to contact in person the URW++ managing director, Dr. Peter Rosenfeld, and, after long negotiations, received the gracious approval for the additional license.

All the fonts released so far are licensed under the GUST Font License (GFL; see [30]), except for TG DejaVu Math which has a somewhat complex license (see [9], the Manifest file for TG DejaVu Math). Recently, many fonts are being released under the SIL Open Font License (OFL; see [34]); therefore, we consider dual-licensing GUST fonts (GFL+OFL).

More details concerning legal matters relevant to GUST e-foundry fonts can be found in a series of publications by Jerzy Ludwichowski — see, for example [18, 19, 20, 21].

7.4. Constraints of tradition vs. our dreams

We are not particularly enthusiastic about font technology nowadays, but we are not going to spit into the wind, and try to make the best of what is available. This does not mean, however, that we are going to relinquish dreams of a successor to the currently prevalent "Knuth – Gutenberg" model of typesetting, that is, the model of stiff rectangles (types) arranged within a larger rectangle (page; a series of pages being called a document), deeply ingrained in Johannes Gutenberg's technology of movable type, and transferred by Donald E. Knuth, among others, to the realm of computers.

Computers facilitate some operations, such as kerning and ligatures, thereby accelerating work on documents. The ease of use of affine transformations is also considered an advantage of computers by many graphic designers. We are inclined to consider both these aspects as being of rather equivocal benefit. Leaving aside these philosophical questions and a far-reaching yet obvious idea, in fact also philosophical, that a font could be defined as a structured collection of general purpose object programs, we confine ourselves to pointing out two examples of aspects that might be improved within the framework of the contemporary edifice of typesetting.

Shrinkable and stretchable spaces, as in TEX, would supposedly be convenient also in OTF fonts and seem not too hard to implement. This simple problem touches on a fairly general and not in the least trivial problem which could be

called “the conflict of competence”: which “knowledge” should be implemented in a font and which — in a typesetting program.

Another improvement, this time hard to implement, is related to a naïve question: why must additional alphabets be embedded into a single math font? The answer is simple: operating systems are poorly designed with regard to font management. In particular, a user is not allowed to define a “private” family of fonts — the commonplace pattern “regular, regular italic, bold, bold italic” is very difficult to dislodge. One can imagine other variants of this idea, namely, borrowing components for extensible characters or kerns from several fonts. As far as we know, such an approach has not been implemented in any typesetting program or any operating system. TEX is no exception (unless virtual fonts are used); note, for example, that changing fonts within a word switches off the TEX hyphenation mechanism.

It should be emphasized that several such problems were tackled in LuaTEX by Hans Hagen and team; it does not seem, however, as if amendments of this kind are to be introduced worldwide in any near future. This is understandable, as the constraints of tradition and compatibility usually slow down the innovative processes. For example, recently announced advancements in the OTF format specification (see, e.g., [6]), can actually be considered a step backward towards a previously abandoned idea, temporarily as it turns out, of *multiple master fonts*. Anyway, this announcement augurs both ill and well for us: it means, on the one hand, that we will probably have to reimplement MetaType 1 in order to keep pace with the surrounding world and, on the other hand, that we still have something to do and something to think about.

8. Acknowledgements

We are indebted to all the people and TEX groups that have supported our font enterprises. Almost all the GUST e-foundry projects have been kindly supported by the Czechoslovak TEX user group $\mathcal{CS}$TUG, the German-speaking TEX user group DANTE e.V., the Polish TEX Users Group GUST, the Dutch-speaking TEX user group NTG, TUG India, UK-TUG, and, last but not least, TUG. In a few cases, GUTenberg, the French-speaking TEX Users Group, supported us too.

We are very grateful to Karel Píška and Ulrik Vieth, who performed extensive tests of our fonts and inspired us with insightful comments, and to Marcin Woliński, who provided the indispensable LATEX support for our fonts.

The exceptional, personal thanks we owe to our friends who kept our spirits up for many years and tirelessly encouraged us to work on fonts: Hans Hagen, Johannes Küster, Jurek Ludwichowski, Volker RW Schaa, Jola Szelatyńska — hearty thanks!

References

Presentations, publications and packages

[1] Lars Engebretsen, *Almost European Fonts*
https://ctan.org/pkg/ae

[2] Michael Everson, *The Alphabets of Europe*
http://www.evertype.com/alphabets/

[3] Lee Hetherington, Eddie Kohler, *T1utils*
http://www.lcdf.org/type/t1asm.1.html
http://www.lcdf.org/type/t1disasm.1.html

[4] John D. Hobby, *MetaPost*
https://ctan.org/pkg/metapost

[5] Donald E. Knuth, *The TEXbook*, *TEX: The Program*, *The Metafontbook*, *Metafont: The Program*, *Computer Modern Typefaces*, Computers & Typesetting, vol. A – E, Addison-Wesley, Reading, Massachusetts, 1986

[6] John Hudson, *Introducing OpenType Variable Fonts*, 2016
https://medium.com/@tiro/https-medium-com-tiro-introducing-opentype-variable-fonts-12ba6cd2369

[7] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *Antykwa Półtawskiego: a parameterized outline font*, Proceedings of the EuroTEX Conference, Heidelberg, Germany, 1999
article:
http://www.staff.uni-giessen.de/partosch/eurotex99/jackowski
download: http://www.gust.org.pl/projects/e-foundry/poltawski

[8] Bogusław Jackowski, Marek Ryćko, *Polish extension of Computer Modern fonts*
https://ctan.org/pkg/pl-mf

[9] Bogusław Jackowski, Piotr Strzelczyk, Piotr Pianowski, GUST e-foundry math fonts
The Latin Modern Math (LM Math) font:
http://www.gust.org.pl/projects/e-foundry/lm-math
The TEX Gyre (TG) Math Fonts:
http://www.gust.org.pl/projects/e-foundry/tg-math
https://ctan.org/pkg/tex-gyre-math

[10] Bogusław Jackowski, Piotr Strzelczyk, *How to make more than one math OpenType font or the Beasts of Fonts*, DANTE 2011 Meeting, Bremen, Germany, 2011
http://www.gust.org.pl/projects/e-foundry/math/beasts05.pdf

[11] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *The Latin Modern (LM) Family of Fonts*
http://www.gust.org.pl/projects/e-foundry/latin-modern

[12] Bogusław Jackowski, Janusz M. Nowacki,
Latin Modern fonts: how less means more
Proceedings of the EuroTEX conference, Pont-à-Mousson, France, 2005
http://tug.org/TUGboat/tb27-0/jackowski.pdf

[13] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *MetaType 1: a MetaPost-based engine for generating Type 1 fonts*, 2001
article: http://www.ntg.nl/maps/26/15.pdf
presentation: http://ntg.nl/eurotex/JackowskiMT.pdf
download: https://ctan.org/pkg/metatype1

[14] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *The TEX Gyre (TG) Collection of Fonts*
http://www.gust.org.pl/projects/e-foundry/tex-gyre

[15] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *TEX Gyre Pagella Math or Misfortunes of Math Typographer*, BachoTEX XX, Bachotek, Poland, 2012
http:
//www.gust.org.pl/projects/e-foundry/math/misfortunes02.pdf

[16] Jörg Knappen, Norbert Schwarz, *European Computer Modern Fonts*
https://ctan.org/pkg/ec

[17] Donald E. Knuth, *Metafont and MetaPost logo fonts.*
https://ctan.org/pkg/mflogo-font

[18] Jerzy B. Ludwichowski, *GUST font licenses*, BachoTEX XIV, Bachotek, Poland, 2006
http://tug.org/fonts/licenses/gfl.pdf

[19] Jerzy B. Ludwichowski, Karl Berry, *GUST Font License: An application of the LATEX Project Public License*, XVII European TEX Conference and BachoTEX XV, Bachotek, Poland, 2007; *TUGboat*, Volume 29 (2008), No. 1
http://www.gust.org.pl/projects/e-foundry/licenses/tb91Berry_
Ludwichowski.pdf

[20] Jerzy B. Ludwichowski, *Licensing of the TEX Gyre family of fonts*, XIX European TEX Conference and 3rd International ConTEXt User Meeting, The Hague, The Netherlands, 2009
https://www.ntg.nl/EuroTeX/2009/slides/jerzy-slides.pdf

[21] Jerzy B. Ludwichowski, *Is there life besides licensing?*, DANTE e.V. General Meeting, Bremen, Germany, 2011
https://www.dante.de/events/Archiv/dante2011/programm/
vortraege/folien-jl.pdf

[22] Janusz M. Nowacki, *Polish fonts*
http://jmn.pl/en/

[23] Piotr Strzelczyk, *Standard Unicode w typografii*, Acta Poligraphica nr 1/2013 (in Polish; to be published also in English under the title *Standard Unicode in typography*)
http://www.cobrpp.com.pl/actapoligraphica/uploads/pdf/AP2013_01_Strzelczyk.pdf
see also: Piotr Strzelczyk, *(uni)coding of math fonts*, BachoTEX XIX, Bachotek, Poland, 2011
http://www.gust.org.pl/bachotex/2011-en/presentations/Strzelczyk_1_2011
[24] Ulrik Vieth, *OpenType math illuminated*, *TUGboat*, Volume 30 (2009), No. 1
http://tug.org/TUGboat/tb30-1/tb94vieth.pdf
[25] George Williams and FontForge project contributors, *FontForge*
http://fontforge.github.io/en-US/

General purpose documentation:

[26] *Adobe Font Development Kit for OpenType*
http://adobe.com/devnet/opentype/afdko.html
[27] *Adobe base 35 fonts*
Adobe original AFM files:
ftp://ftp.adobe.com/pub/adobe/type/win/all/afmfiles/base35/
URW replacement: https://ctan.org/pkg/urw-base35
[28] *Adobe Type 1 Font Format*
https://partners.adobe.com/public/developer/en/font/T1_SPEC.PDF
[29] *MATH — The mathematical typesetting table*
https://www.microsoft.com/typography/OTSPEC/math.htm
[30] *GUST Font License*
http://www.gust.org.pl/fonts/licenses/GUST-FONT-LICENSE.txt
http://tug.org/fonts/licenses/GUST-FONT-LICENSE.txt
[31] *OpenType specification (full)*
http://www.microsoft.com/en-ph/download/details.aspx?id=1144
[32] *OpenType Feature File Specification*
http://www.adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html
[33] *Registered features — definitions and implementations (p – t)*
https://www.microsoft.com/typography/otspec/featuretags.htm
[34] *SIL Open Font License*
https://scripts.sil.org/OFL
[35] *The Unicode Standard 9.0.0, 2016*
http://unicode.org/versions/Unicode9.0.0

[36] *The Unicode Standard: A Technical Introduction*
`http://unicode.org/standard/principles.html`
[37] Barbara Beeton, Asmus Freytag, Murray Sargent III, *Unicode Technical Report #25. Unicode Support for Mathematics*
`http://unicode.org/reports/tr25/`

All links above were tentatively accessed on 5th July 2015.

Fontové projekty e-písmolijny GUST

V tomto článku shrnujeme naši práci na TEX Gyre fontech. Také popisujeme nesrovnalosti v existujících fontech a kódováních a ukazujeme, jak je toto vyřešeno v TEX Gyre fontech.

Klíčová slova: TEX Gyre fonty, LM fonty, AFDKO, e-písmolijna GUST.

Bogusław Jackowski – Gdańsk, Poland
b_ jackowski@gust. org. pl

Piotr Strzelczyk – Sopot, Poland
p. strzelczyk@gust. org. pl

Piotr Pianowski – Trąbki Wielkie, Poland
p. pianowski@gust. org. pl

The SWIGLIB Project

Luigi Scarso

The SWIGLIB project aims to show a way to build and distribute shared libraries for LuaTEX by means of SWIG. This paper depicts the infrastructure that has been created and the rationale behind it. Simple examples are shown.

Keywords: LuaTEX, SWIGLIB, SWIG, external libraries

Introduction

The Lua language is well-known for its simplicity and compactness, and also for its easy integration into an existing project. This integration refers both to compilation—TEX Live currently provides binaries for 21 platforms and all of them have a `luatex` executable—and in a more general sense the relatively small amount of time required to get acquainted with its constructs and data structures.

Analogous to the `\usepackage` macro of LATEX, it is easy to extend the built-in features of Lua by means of external Lua modules, usually loaded with `load("⟨module_name⟩")`. What perhaps is less well known is that the same is also available for *binary* modules; for example, a C library compiled in the native format of the platform. This is due to the double nature of Lua, as both an interpreted language and a library that can be linked with an application (see Ierusalimschy, 2013, p. 249):[1] the interaction of the Lua library and the application must follow the application programming interface (API) of Lua.

While for LuaTEX there is currently no official C API—it is a program, not a library—the Lua API is completely described in the Lua manual (`http://www.lua.org/manual`) and it counts 245 items, including constants, macros, functions and standard libraries. They consistently use a stack to exchange data (and hence several functions are dedicated to the stack manipulation) and use an opaque data structure to store the current state, but the stack is accessible only by the state and sometimes it is confused with it. By design (related to the choice of ANSI C for the implementation language) the Lua state is not thread-safe, but the library is carefully designed to avoid destructive interference in global variables and in some cases multithreading with a single shared state appears to be possible (Scarso, 2014). In any case, the best solution is to avoid sharing the

[1] By design the standard Lua library is written in ANSI C and it is precisely for this reason that integration into disparate platforms is easy.

doi: 10.5300/2016-1-4/47

```
-- a Lua function that adds two numbers
function add ( x, y )
  return x + y

/* The C code that calls the Lua function; */
/* we suppose that the state L             */
/* is already initialised.                 */

/* Lua headers */
#include <lua.h>

int lua_add ( int x, int y ){
 int sum;
 lua_getglobal(L, "add");  /* function name   */
 lua_pushnumber(L, x);     /* first argument  */
 lua_pushnumber(L, y);     /* second argument */
 lua_call(L, 2, 1);        /* call the function
            with 2 arguments, return 1 result */
 sum = (int)lua_tointeger(L, -1);/* get result*/
 lua_pop(L, 1);            /* clear the stack */
 return sum;               /* return the sum  */
}
```

Figure 1: Calling a Lua function from C.

state between multiple threads—the library can in fact safely manage different states, at the price of more complex code.

Every "well done" C library exposes its services by means of an API which is, of course, completely unrelated to the Lua API. Communication between the two can happen in either direction: when the application library wants to execute a Lua function it has to follow the Lua API as shown for example in Figure 1, and similarly when a C function is called by the Lua interpreter (see Figure 2)—and this latter case is the subject of this paper. It is clear that if an application library has tens or hundreds of functions, writing the corresponding code can take a considerable amount of time.

Before discussing the tools and the infrastructure used, it is worth mentioning at least these three scenarios where an application library can be useful:

- *pre/post processing* of data, typically pre- processing images (i.e. conversion) and post- processing PDFs;
- *extending* LuaTEX, for example to connect to a database at runtime;
- *extending* the application with LuaTEX as a scripting language—probably a less common, but still important, use.

```
/* Example C function to be called from Lua. */

/* Lua headers  */
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>

int c_add (int x, int y) {
 return x+y;
}

int _wrap_c_add (lua_State *L) {
 int x,y, sum;
 x = (int)lua_tointeger(L, -1); /* first arg   */
 y = (int)lua_tointeger(L, -2); /* second arg  */
 sum = c_add(x,y);              /* call c_add  */
 lua_pushnumber(L, sum);        /* push result */
 return 1;                      /* return sum  */
}

static const luaL_Reg myapplication [] = {
  {"add", _wrap_c_add},  /* register c_add */
  {NULL,NULL}            /* sentinel       */
};

int luaopen_myapplication(lua_State *L) {
 luaL_newlib(L,myapplication);
 return 1;
}

-- Calling c_add from Lua
local myapplication = require("myapplication")
print (myapplication.add(2,3))
```

Figure 2: Calling a C function from Lua.

The SWIG tool

As described above, to connect an application library with the Lua interpreter a third layer which acts as interface is needed. This layer, called *wrapper code*, must know the application API and, of course, the Lua API. In Figure 2, `c_add` is the application function, and the wrapper code items are `_wrap_c_add`, `myapplication` and `luaopen_myapplication`; the local Lua variable `myapplication` is the *binding*. Under Linux the compilation is straightforward:

```
$ gcc -I/usr/include/lua5.2 -fPIC \
  -o myapplication.o \
  -c myapplication.c
$ gcc -I/usr/include/lua5.2 -shared \
  -o myapplication.so myapplication.o \
  -llua5.2
```

where `-fPIC` tells the compiler to generate position independent code, given that `myapplication.so` is a shared library. From this elementary example we can identify the following issues:

- how to generate a wrapper for a rich and complex application API?
- how to compile the wrapper to obtain a suitable binary module?
- how to distribute the module?

The next subsections will try to address these questions.

Generate a wrapper

After a initial period of experimentation the following assumptions have emerged as suitable for a project that aims to serve the TEX community:

1. the wrapper code should be generated in an automatic fashion preserving as much as possible the meaning and the names of the functions and data structures of the application API;
2. the application and Lua API should be freely accessible.

The tool chosen is SWIG, the *Simplified Wrapper and Interface Generator* program available for different platforms, including Linux, Windows and MacOSX. Its web site is `http://www.swig.org`; for a quick overview, see also `http://www.ibm.com/developerworks/aix/library/au-swig`. SWIG has a powerful C/C++ preprocessor and can analyse[2] a header file and produce the wrapper code. For example, given the C API

[2]SWIG works particularly well with C libraries, while with C++ libraries usually the developer has to manually write some customisation, e.g. to manage function overloading or multiple inheritance. For C++, in fact, "at the lowest level, SWIG generates a collection of procedural ANSI C-style wrappers"; see `http://www.swig.org/Doc3.0/SWIGDocumentation.html#SWIGPlus_nn2`.

```
/* myapplication.h */
#include <lua.h>
#include <lauxlib.h>
#include <lualib.h>

extern int c_add (int, int);
```

the SWIG interface file to create the wrapper is:

```
%module core
%{
/* code included in the wrapper */
#include "myapplication.h"
%}
/* header to analyse */
%include "myapplication.h"
```

The wrapper itself (by default `core_wrap.c`) is generated with

```
$ swig -lua core.i
```

and, supposing that the application header and the shared library `myapplication.so` live in the current directory `./`, the binary module `core.so` is compiled as below (again for the Linux platform):

```
$ gcc -I/usr/include/lua5.2 -I./ -fPIC \
      -o core_wrap.o \
       -c core_wrap.c
$ gcc -L./ -Wl,-rpath,'$ORIGIN/.' -shared \
      -o core.so core_wrap.o \
      -lmyapplication -llua5.2
```

and loaded in Lua with

```
local myapplication = require("core")
print (myapplication.c_add(2, 3))
```

This example shows all the basic components used in the SWIGLIB project. A practical interface file is in fact only a bit more complex: here is one for the cURL library, a free and easy-to-use client-side URL transfer library (`http://curl.haxx.se/libcurl`):[3]

```
%module core
#ifdef SWIGLIB_WINDOWS
%include <windows.i>
#endif

/* Section for utilities, such as  */
/* built-in wrappers for C arrays, */
/* C pointers, function pointers.  */
...
```

[3]The real file has a few more directives, but this example shows the important pieces.

```
/* API */
%{
#include "curl/curl.h"
%}

/* Headers to generate the wrapper */
%include "curl/curlver.h"
%include "curl/curlbuild.h"
%include "curl/curlrules.h"
%include "curl/curl.h"
%include "curl/easy.h"
%include "curl/multi.h"

/* Customisation   */
%include "native.i"
%include "inline.i"
%include "luacode.i"
```

Each binary module of the SWIGLIB project is named `core`, so each needs to be saved into a specific directory, as will be shown later. Next, there is a section to eventually include the wrappers that SWIG supplies by default for the basic C types such as `char`, `int`, `long`, etc. (useful, for example, when a parameter of a function is an array or a pointer to a basic type). After that is the section that includes the application API into the wrapper and generates the wrapper; the order of the `%include` directives is not random, but reflects the dependencies between the headers.[4] Finally, the `native.i` file is used when the developer wants to replace the standard SWIG wrapper of a function with a custom implementation; the `inline.i` file is useful to add new members to the application API; and the `luacode.i` file to add Lua code when the module is initialised at loading time.

Normally, these `.i` files are empty but it turns out that our example of the cURL API has several functions that take a variably-typed argument—either a pointer to a long or a pointer to a char, etc.; in any case, a finite set of types as described in the documentation of the API. Here the `inline.i` file defines, for each variation of such functions, several C helper functions with the third argument fixed; i.e. one function with a pointer to a long, a second with a pointer to a char, etc. The `luacode.i` file has the single Lua function that calls the helper functions with the right third argument: of course this means that a lot of code is hand-written, given that a single function can have 3 or 4 helper functions—it sounds complicated but it is not especially difficult.[5]

[4] `gcc -H` can be used with a header file to print out its dependencies.

[5] Although the chapter "Variable Length Arguments" at

In most cases this simple organisation of the interface file is enough, but it can be extended in two ways: first, to build a *helper* module that consists solely of SWIG directives as in

```
%module core
#ifdef SWIGLIB_WINDOWS
%include <windows.i>
#endif
%include "carrays.i"
%include "cpointer.i"
%include "constraints.i"
%include "cmalloc.i"
%include "lua_fnptr.i"

%{ /* array functions */ %}
%array_functions(char, char_array);
%array_functions(unsigned char, u_char_array);
%array_functions(char*, char_p_array);
%array_functions(unsigned char*, u_char_p_array);
/* Several other SWIG directives ...*/
```

Second, by adding C functions and data structures to the inline interface a user can build a custom *usermodule*, eventually using other application libraries. In other words, SWIG also supports interface files `usercore.i`, `usernative.i`, `userinline.i` and `userluacode.i` and hence a `usercore` binary module that stays in the same directory as the `core` application.

Compilation of a wrapper

Compilation of binary modules is not as difficult as it seems at first sight: given an application header and the *corresponding* shared library, SWIG generates ANSI C wrapper code, which is usually both portable and easily compilable. Of course much depends on the application library, but currently all the modules provided are compiled for 64-bit Linux (Ubuntu 14.04 LTS) with the GCC toolchain and cross-compiled for Microsoft Windows 32-bit and 64-bit using the Mingw-w64 toolchain; it is also possible under Linux to use the native compiler suite for Windows from `http://tdm-gcc.tdragon.net`

In this way the application headers and library match among different platforms (only two in this phase) which in turn means that at the LuaTEX level the interface to the application library is the same. While the compilation of an application module almost always uses the `configure` script generated from the GNU Autotools, SWIGLIB uses for the wrapper simple `bash` scripts; for example, for `curl` under Linux:

`http://www.swig.org/Doc3.0/SWIGDocumentation.html#Varargs` does start with *a.k.a. "The horror. The horror."*

```
trap 'echo "Error on building library"; exit $?' ERR
echo "building for      : linux 64bit"
## SWIG
swig -I$(pwd)/resources/include64 -lua -o core_wrap.c ../core.i
## Compile wrapper
rm -f core_wrap.o
gcc -O3 -fpic -pthread -I$LUAINC -I./resources/include64/ \
    -c core_wrap.c -o core_wrap.o
## Build library
rm -f core.so
CFLAGS="-g -O3 -Wall -shared -I./resources/include64
        -L./resources/lib64"
LIBS="-lcurl -lssh2"
gcc $CFLAGS -Wl,-rpath,'$ORIGIN/.' core_wrap.o $LIBS -o core.so
## End
mv core.so resources/lib64
rm core_wrap.o
rm core_wrap.c
```

and for Windows 64-bit it is almost the same:

```
trap 'echo "Error on building library"; exit $?' ERR
## SWIG
swig -DSWIGLIB_WINDOWS -I$(pwd)/resources/include64 \
     -lua -o core_wrap.c ../core.i
## Compile the wrapper
rm -f core_wrap.o
$GCCMINGW64 -O3 -I$LUAINC -I./resources/include64/ \
   -c core_wrap.c -o core_wrap.o
## Build the library
rm -f core.dll
CFLAGS="-O3 -Wall -shared "
LIBS="$LUALIB/$LUADLL64
      resources/lib64/libssh2-1.dll
      resources/lib64/zlib1.dll
      resources/lib64/libcurl-4.dll
      resources/lib64/ssleay32.dll
      resources/lib64/libeay32.dll "
$GCCMINGW64 $CFLAGS \
    -Wl,-rpath,'$ORIGIN/.' core_wrap.o $LIBS -o core.dll
## End
mv core.dll resources/lib64
rm core_wrap.o
rm core_wrap.c
```

A simple bash script should be easily ported to different platforms: the GNU Autotools are well suited for Unix-like systems, but Windows has its own toolchain

and such a shell script can be translated in a batch script without much effort, giving a good starting point (see Calcote, 2010, p. 3).

A binary module can easily depend on other binary modules. Under Windows, these modules are searched first in the same directory of the wrapper, but in Linux (and hopefully on other Unix-like systems too) that "local" search is enforced with the linker option `-Wl,-rpath,'$ORIGIN/.'`. We do this to keep the wrapper module and its dependencies as much as possible self-contained in a TDS tree.

In spite of the efforts to mask the differences between the systems, at some point they emerge and it is not always possible to find a nice way to manage them. One of these differences is symbol resolution and collision: when an application module has a reference to an external symbol (i.e. a function or a data item), under Linux this reference is resolved at run-time while in Windows it must be resolved at build-time, when the module is compiled.

Given that an application module always needs to resolve the Lua API symbols, the first consequence is that the `luatex` Windows binary must be compiled with a dynamic link to an external Lua library (a Lua DLL) and the same DLL must be used at build-time for the application module. Under Linux the Lua API symbols are unequivocally resolved inside the `luatex` binary, but if the application module needs a symbol from another API (for example, a function from `libpng`, which is part of `luatex`) it must resolve that symbol to an external auxiliary library and not inside the `luatex` binary: with Windows this happens automatically because, by default, the symbols are not visible if not explicitly marked as such, but in Linux the situation is just the opposite. The `luatex` binary must be compiled with the `gcc` flag `-fvisibility=hidden`—this will be the default starting with TeX Live 2015:[6] hence, all the Linux binaries before this date are not safe.

Another fundamental difference is that Linux 64-bit and Windows 64-bit do not use the same data model. Linux uses the so-called LP64, where the type `long` and a pointer are both 64 bits, while Windows uses LLP64, where a `long` is 32 bits and a pointer 64 bits. As a consequence, if a program under 64-bit Linux uses a `long` to store an address, it cannot be automatically ported to 64-bit Windows. Although the 64-bit Windows version could use the type `long long`, this is a C99 extension and it is not supported by the Microsoft Visual C compiler. The situation is no better in C++: the following example that uses GMP 6.0.0 fails to compile with Mingw-w64 but works with GCC under Linux[7]—and in both cases `sizeof a` returns 8.

```
#include <gmpxx.h>
#include <iostream>
using namespace std;
```

[6] Peter Breitenlohner has done incalculable work in implementing the symbol visibility and the build of shared versions of the TeX-specific libraries.

[7] And the fork MPIR 2.7.9 compiles correctly under Mingw-w64 and gives the same result as Linux!

```
int main(void) {
    size_t a = 5;
    mpz_class b(a);
    cout << b.get_ui() << endl;
    cout<< sizeof a <<endl;
    return 0;
}
```

Deployment

The SWIGLIB project is hosted[8] at `http://swiglib.foundry.supelec.fr` with a public readonly Subversion source repository accessible at `http://foundry.supelec.fr/projects/swiglib`. The root has currently the following application modules:

```
trunk
├─ attic
├─ basement
├─ curl
├─ experimental
├─ ghostscript
├─ graphicsmagick
├─ helpers
├─ leptonica
├─ libffi
├─ lua
├─ luarepl
├─ mysql
├─ parigp
├─ physicsfs
├─ postgresql
├─ qpdf
├─ R
├─ sqlite
├─ swig
├─ usermod
└─ zeromq
COPYRIGHT
build.sh
```

Each application module has the following layout (here shown for `curl`):

```
curl
└─ 7.40.0
```

[8] Thanks to Fabrice Popineau for his invaluable support.

```
├─ docs
├─ linux
│  ├─ resources
│  │  ├─ include32
│  │  ├─ include64
│  │  │  └─ curl
│  │  ├─ lib32
│  │  └─ lib64
│  └─ test
│     build-linux-x86_64.sh
├─ osx
│  └─ resources
│      ├─ include32
│      ├─ include64
│      ├─ lib32
│      └─ lib64
└─ windows
    ├─ resources
    │  ├─ include32
    │  │  └─ curl
    │  ├─ include64
    │  │  └─ curl
    │  ├─ lib32
    │  └─ lib64
    └─ test
    build-mingw32.sh
    build-mingw64.sh
    build-msys32.sh
    build-msys64.sh
core.i
inline.i
luacode.i
native.i
```

where `lib64` (`lib32`) hosts the application API and `lib64` (`lib32`) the binary module. The `osx` directory is a placeholder—currently it is empty. The `lua` directory contains the Lua API and the binaries for Linux 64-bit, Windows 32-bit and 64-bit:

```
└─ luatex-beta-0.79.3.1
    ├─ include
    │  ├─ lauxlib.h
    │  ├─ luaconf.h
    │  ├─ lua.h
    │  ├─ lua.hpp
    │  ├─ lualib.h
```

```
│   ├─ patch-01-utf-8
│   ├─ patch-02-FreeBSD
│   └─ patch-03-export
├─ linux
│   └─ luatex
├─ w32
│   ├─ libkpathsea-6.dll
│   ├─ luatex.exe
│   └─ texlua52.dll
└─ w64
    ├─ libkpathsea-6.dll
    ├─ luatex.exe
    └─ texlua52.dll
```

Application module location in the TDS
The natural location of a binary module inside a TDS directory is under `bin/`. The current layout looks like the following (for Linux 64-bit):

```
tex
└─ texmf-linux-64
    └─ bin
        └─ lib
            └─ luatex
                └─ lua
                    └─ swiglib
                        └─ curl
                            └─ 7.40.0
                                ├─ core.so
                                └─ libcurl.so
```

SWIGLIB doesn't require a particular method to load a wrapper module, because this is a task of the format. The tests in the Subversion repository use the low-level Lua function `load`, but they need to know the system and the full path of the module; on the other hand, ConTeXt has a global `swiglib` function (see `util-lib.lua` and Hagen, 2016) that is independent from the system and the path—but it doesn't use the `kpse` library.

Conclusions

Without a doubt, building a wrapper module requires a working knowledge at least of the C language, for which Kernighan and Ritchie (1988) is still a pleasure to read; useful information on shared libraries is also in Drepper (2011) and Levine (1999) while for Linux (Kerrisk, 2010) is still one of the best references, as Russinovich (2012a) and Russinovich (2012b) are for Windows.

Moreover, having a working wrapper is only half of the story: the rest is a working Lua/TEX layer that suits with the format in use—and this cannot be part of the underlying SWIGLIB. The example with GMP 6.0.0 shows that an application module that compiles well and passes all the tests can still fail to compile an apparently innocuous program. The C code itself is not always easy to understand, as for example with the following program

```
/* test.c    */
#include <stdlib.h>
void foo(int *x){
  int y = *x;
  if (x == NULL)
    return;
}
int main(){
  int *x;
  x = NULL;
  foo(x);
  return 0;
}
```

which gives a segmentation fault if compiled with `gcc` without optimisation (`gcc -o test test.c`), but it is ok with optimisation (`gcc -O3 -o test test.c`).[9] Portable multithreading also looks problematic, due to the lack of support in ANSI C and hence in Lua. Of course the Linux and Windows platforms are not the only ones to consider and the absence of MacOSX is the most notable; FreeBSD as well, which seems to be rather easier to add.

Despite these issues, SWIG is an exceptionally flexible program, and it can adapted to manage almost any situations. If an interface file is complicated, it can often be simplified with an auxiliary C module; if a user needs to customise an application module, this can be done by adding a set of Lua functions and/or C functions—and all this while always formally writing an interface file. A possible objection is that LuaTEX does not have a read-eval-print loop ("repl") program as standard Lua does, but SWIGLIB has a pure Lua module `luarepl` that mimics the original one quite well. This means that it is possible to use LuaTEX as a general-purpose scripting language, i.e. to manage the installation of TEX packages.

Regarding LuaJITTEX (Scarso, 2013): even when it is possible to use the same interface file, the API and the `luajitex` libraries are not the same. Furthermore, LuaJIT users seem to prefer the use of the LuaJIT `ffi` module, which is roughly similar to SWIG. It should still be doable to implement in SWIG via a new backend LuaJIT-ffi that emits `ffi` chunks instead of the LuaJIT C

[9] `y = *x` results in undefined behaviour when `x` is `NULL`, but the optimisation `-O3` is able to detect that `y` is never used and it deletes it.

API, effectively eliminating the need for a C compiler. Clearly work for the future.

Some practical examples of applications are shown in Hagen (2016) and Scarso (2011). These will also be the subject of a future paper.

References

Calcote, John. *Autotools: A Practitioner's Guide to GNU Autoconf, Automake, and Libtool.* 1. ed. San Francisco : No Starch Press, 2010. xxiv + 332 pp. ISBN 978-1-59327-206-7.

Drepper, Ulrich. *How to Write Shared Libraries* [on-line]. 2011. [cit. 2015-03-05]. Available at: `http://www.akkadia.org/drepper/dsohowto.pdf`.

Hagen, Hans. *SWIGLIB basics* [on-line]. 2016. [cit. 2016-12-25]. Available at: `http://www.pragma-ade.com/general/manuals/swiglib-mkiv.pdf`.

Ierusalimschy, Roberto. *Programming in Lua.* 3. ed. Rio de Janeiro : Lua.Org, 2013. 366 pp. ISBN 978-85-903798-5-0.

Kernighan, Brian W.; Ritchie, Dennis M. *The C Programming Language.* 2. ed. Englewood Cliffs (NJ) : Prentice Hall, 1988. xii + 272 pp. ISBN 0-13-110370-9.

Kerrisk, Michael. *The Linux Programming Interface: A Linux and UNIX System Programming Handbook.* 1. ed. San Francisco : No Starch Press, 2010. 1552 pp. ISBN 978-1-59327-220-3.

Levine, John R. *Linkers and Loaders.* 1. ed. San Francisco : Morgan Kaufmann Publishers, 1999. 256 pp. ISBN 1-5586-0496-0.

Russinovich, Mark E., Solomon, David A., Ionescu, Alex. *Windows Internals, Part 1: Covering Windows Server 2008 R2 and Windows 7.* 6. ed. Redmond : Microsoft Press, 2012a. 752 pp. ISBN 978-0-7356-4873-9.

Russinovich, Mark E.; Solomon, David A.; Ionescu, Alex. *Windows Internals, Part 2: Covering Windows Server 2008 R2 and Windows 7 (Windows Internals).* Redmond : Microsoft Press, 2012b. 672 pp. ISBN 978-0-7356-6587-3.

Scarso, Luigi. Extending ConTeXt MkIV with PARI/GP. *ArsTEXnica*, 2011, Vol. 11, p. 65–74. (ISSN 1828-2369.)

Scarso, Luigi. LuaJITTEX. *TUGboat*, 2013, Vol. 34, No. 1, p. 64–71. (ISSN 0896-3207.)

Projekt SWIGLIB

Článek se věnuje tématu přístupu k binárním knihovnám v LuaTeXu. Popisuje problémy s přístupem ke sdíleným knihovnám na rozdílných platformách a tvorbou rozhraní zpřístupňujících tyto knihovny pro programy v jazyce Lua.

Pro zjednodušení tvorby rozhraní pro binární knihovny je navrhnut projekt SWIG, který umožňuje jejich poloautomatickou tvorbu za pomoci konfiguračního souboru a analýzy hlavičkových souborů zpracovávaných knihoven. Projekt SWIGLIB pak řeší způsob kompilace tohoto rozhraní na rozdílných platformách a umístění výsledných binárních knihoven tak, aby byly přístupné LuaTeXu. Pro nahrání knihoven je třeba upravit nahrávací rutiny v Lue. Existuje podpora v ConTeXtu, pro ostatní formáty musí podpora teprve vzniknout.

Klíčová slova: LuaTEX, SWIGLIB, SWIG, externí knihovny

In fulfillment of the TEX Development Fund grant no. 23, Dynamic library support in LuaTEX, 2013. Grants from (in alphabetical order) CSTUG, DANTE *e.V.,* GUST, NTG *and* TUG.

Luigi Scarso, `luigi.scarso@gmail.com`

Dobře dokumentované statistické výpočty

Marek Pomp

Článek popisuje implementaci metody dobře dokumentovaných programů v prostředí pro statistické výpočty. Speciálně se věnuje balíčku `Sweave` určeném pro psaní dynamických dokumentů, jejichž výpočty se provádějí za pomocí statistického programu R.

Klíčová slova: literární programování, reprodukovatelný výzkum, Sweave, jazyk R.

Úvod

V roce 1981 Donald E. Knuth formuloval svou ideu literárního programování, na jejímž principu bylo vytvořeno mnoho nástrojů usnadňujících vytváření dobře dokumentovaných programů. Tento trend se dotknul i systémů pro statistické výpočty, viz např. (Gandrud, 2013).

V posledních letech začali statistikové v široké míře používat statistický program R, volně šiřitelnou implementaci jazyka S. V roce 2002 Friedrich Leisch vytvořil systém `Sweave`, který umožňuje kombinovat LaTeXovský text se zdrojovým kódem, který je vyhodnocen programem R, viz (Leisch, 2002) a (Leisch, 2008).

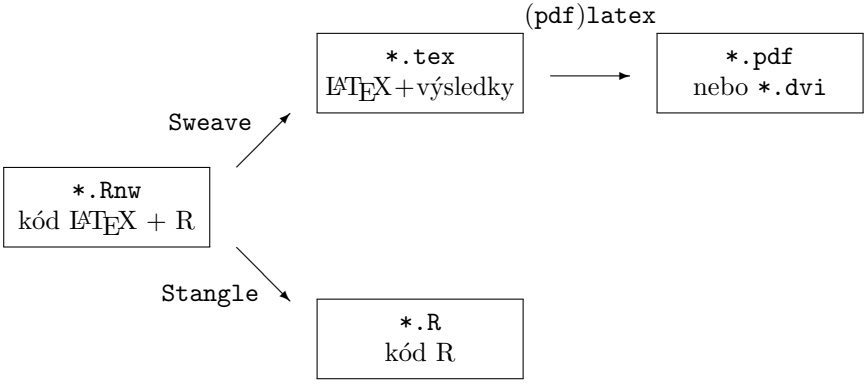
V roce 2012 navázal na balíček `Sweave` Yihui Xie svým programem `knitr`, viz (Xie, 2013), ve kterém obohatil základní možnosti `Sweave` a integroval výpočty programu R také do dalších značkovacích jazyků, jako je HTML nebo Markdown.

`Sweave`, principy

Základem při práci s balíkem `Sweave` je kombinace LaTeXovského zdrojového souboru s výpočty a procedurami programovacího jazyka R, podle pravidel `noweb` (viz Ramsey, 1994). Směs textu pro LaTeX a zdrojového kódu pro R je obvykle uložena v souboru s příponou `Rnw`. Zdrojový soubor `Rnw` se nejprve předloží programu R. R jako preprocesor může mít dvě funkce. Buď to vyhodnotí všechny výpočty a požadované výsledky převede na sekvence, kterým rozumí TeX. Výsledek práce programu R je čistý soubor určený pro zpracování LaTeXem, tomuto procesu říkáme *weave*. Druhou variantou je, že ze souboru `Rnw` je extrahován jen zdrojový text pro program R, tzv. proces *tangle*[1] (viz obr. 1).

[1] Proces *tangle* spouští funkce `Stangle`, která ze souboru `Rnw` vybírá jen kód určený programu R a ukládá jej do souboru s příponou `R`.

doi: 10.5300/2016-1-4/62



Obrázek 1: Schéma práce se `Sweave` a `Stangle`

Kompilace souboru `*.Rnw` (viz obr. 2) v programu R provádí funkce `Sweave`, která do souboru s příponou `tex` beze změny opíše všechen kód určený pro TEX. Kód určený pro program R nechá vyhodnotit a zapíše do stejného souboru výsledky tohoto vyhodnocení spolu s dalšími TEXovými formátovacími příkazy.[2]

```
> Sweave("prvni.Rnw")
Writing to file prvni.tex
Processing code chunks with options ...
 1 : echo keep.source term verbatim (prvni.Rnw:5)

You can now run (pdf)latex on ‘prvni.tex’
```

Výsledný soubor (na obr. 3) je možné běžným způsobem zpracovat pomocí LaTeXu. Prostředí `Schunk`, `Sinput` a `Soutput`, která formátují výstupy, jsou definována v balíku `Sweave.sty`, který je součástí běžných distribucí LaTeXu.

Funkci `Sweave` a následně LaTeX můžeme spouštět odděleně, `Sweave` v prostředí R a následně TEX v operačním systému. Vhodnější ovšem je oba kroky spojit dohromady. Jedna z možností, je spouštět obojí na příkazovém řádku v operačním systému, např. v Linuxu[3] pomocí příkazů

```
$ Rscript -e "Sweave('prvni.Rnw')"; pdflatex prvni
```

nebo v ekvivalentní formě

```
$ R CMD Sweave prvni.Rnw; pdflatex prvni
```

Pomocí předchozích příkazů lze vytvářet složitější skripty a Makefile soubory.

[2] Příkazovou řádku programu R značíme promptem `>`.

[3] Příkazovou řádku operačního systému značíme promptem `$`.

prvni.Rnw

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\begin{document}
Výpočet aritmetického průměru z posloupnosti čísel.
<<echo=TRUE, results=verbatim>>=
x <- c(1,2,3,4)
mean(x)
@
\end{document}
```

Obrázek 2: Zdrojový soubor `prvni.Rnw`, kód pro LaTeX opatřený blokem s kódem pro program R

prvni.tex

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage{Sweave}
\begin{document}
Výpočet aritmetického průměru z posloupnosti čísel.
\begin{Schunk}
\begin{Sinput}
> x <- c(1,2,3,4)
> mean(x)
\end{Sinput}
\begin{Soutput}
[1] 2.5
\end{Soutput}
\end{Schunk}
\end{document}
```

Obrázek 3: Výsledek práce preprocesoru `Sweave`, soubor `prvni.tex`

prvni.pdf

Výpočet aritmetického průměru z posloupnosti čísel.

```
x <- c(1,2,3,4)
mean(x)
```
[1] 2.5

Obrázek 4: Výsledek zpracování souboru `prvni.tex` pomocí LaTeXu, výstup v DVI nebo PDF

Další možností je spouštět `Sweave` i LaTeX z příkazové řádky programu R. Ke spuštění TeXu slouží utilitka `texi2dvi` z balíku `tools`, která spouští LaTeX a BibTeX až do správného vyhodnocení všech křížových odkazů.

```
> Sweave("prvni.Rnw")
> tools::texi2dvi("prvni.tex", pdf=TRUE)
```

V neposlední řadě je možné využít k editaci a následné kompilaci některé z komplexních prostředí pro editaci TeXtu. Styl pro `Sweave` je podporován ve většině textových editorů jako je např. Emacs, vim, WindEdt nebo TeXnicCenter. Samozřejmostí je podpora pro LyX. Za zmínku stojí také R-Studio (viz RStudio Team, 2015), propracovaný nástroj primárně určený pro práci s programem R a vytváření dokumentů obsahujících kód jazyka R.

U souborů s jiným kódováním než ASCII rozpozná `Sweave` typ kódování z LaTeXovského `\usepackage[...]{inputenc}`, ale je možné kódování vnutit funkci `Sweave` parametrem `encoding`, např.

```
$ Rscript -e "Sweave('prvni.Rnw', encoding='utf8')"
```

Blok kódu a jeho základní parametry

Do LaTeXovského zdroje se části určené programu R vkládají jako tzv. *chunks* — *bloky kódu*. Podle zvyklostí v `noweb` je blok kódu na začátku omezen řádkem obsahujícím posloupnost znaků `<<`*parametry*`>>=` a na konci řádkem s jediným znakem `@`. Parametry v hlavičce bloku určují jeho další chování, obvykle formátování výstupu. Příklad zdrojového souboru s jedním blokem kódu pro R je na obr. 2. Tento blok má dva parametry `echo=TRUE` a `results=verbatim`. Oba parametry jsou v tomto příkladu nadbytečné, protože uvedené hodnoty parametrů jsou ve `Sweave` přednastaveny a chování bloku by bylo stejné, jako kdyby byla hlavička prázdná `<<>>=`.

Pro ortodoxní LaTeXisty je možné blok R-kovského kódu uzavřít do prostředí `Scode`.

`\begin{Scode}{`*parametry*`}`
kód pro R
`\end{Scode}`

Aby `Sweave` správně rozpoznal syntaxi pro zápis bloků, měl by soubor s bloky vymezenými prostředím `Scode` mít příponu `Rtex`. Typ zápisu pro bloky lze měnit průběžně v jednom souboru. Změnu syntaxe oznámíme příkazem

`\SweaveSyntax{SweaveSyntaxLatex}`

nebo

`\SweaveSyntax{SweaveSyntaxNoweb}`

Popišme ty nejužívanější parametry pro blok R-kovského kódu. (Hodnoty parametrů jsou uvedeny v závorce, první z uvedených hodnot je ve `Sweave` standardně přednastavena. V příkladech je vlevo blok, tak jak je zapsaný v souboru `*.Rnw` a vpravo výstup ve finálním `*.pdf` souboru.)

`echo` (`TRUE`/`FALSE`) — ukáže/skryje zdrojový kód pro R.

```
<<>>=
x <- c(1,2,3,4)
mean(x)
@
```

```
> x <- c(1,2,3,4)
> mean(x)
[1] 2.5
```

```
<<echo=FALSE>>=
x <- c(1,2,3,4)
mean(x)
@
```

```
[1] 2.5
```

`results` (`verbatim`/`tex`/`hide`) — vypíše výsledky výpočtů buď'to v prostředí `Soutput`, nebo (`results=tex`) výsledky vloží jako čistý text bez jakéhokoliv prostředí, přesně tak, jak je vypisuje program R (vhodné např. pro předformátované tabulky viz str. 73), anebo nevypisuje žádný výsledek (`results=hide`).

```
<<results=hide>>=
x <- c(1,2,3,4)
mean(x)
@
```

```
> x <- c(1,2,3,4)
> mean(x)
```

`label` (text) — jeden z parametrů může být návěští pro blok. Návěští slouží k odkazům na obsah bloku na jiném místě dokumentu, nebo jako název pro obrázky v bloku vytvořené a pod. Odkaz na blok se vytváří posloupností znaků `<<`*label*`>>`.

Je-li `label` uveden jako první z parametrů, můžeme vynechat klíčové slovo `label` a psát jen samotné návěští.

```
<<a, echo=FALSE, results=hide>>=
x <- c(1,2,3,4)
mean(x)
@
```

```
<<>>=
1+1
<<a>>
@
```

```
> 1+1
[1] 2
> x <- c(1,2,3,4)
> mean(x)
[1] 2.5
```

`keep.source` (`FALSE`/`TRUE`) — v základním nastavení `Sweave` upravuje kód pro R tak, že vynechá komentáře a změní řádkování a odsazení. V případě `keep.source=TRUE` přepisuje instrukce pro R přesně tak, jak jsou formátovány ve zdrojovém textu.

```
<<keep.source=FALSE>>=
x <- c(1,   2,
       3,4)
    mean(x) # výpočet
@
```

```
> x <- c(1, 2, 3, 4)
> mean(x)
[1] 2.5
```

```
<<keep.source=TRUE>>=
x <- c(1,   2,
       3,4)
    mean(x) # výpočet
@
```

```
> x <- c(1,   2,
+        3,4)
>     mean(x) # výpočet
[1] 2.5
```

`eval` (`TRUE`/`FALSE`) — povolí/zakáže vyhodnocení zdrojového kódu v R.

`split` (`FALSE`/`TRUE`) — uloží výstupy každý zvlášť do jednotlivých souborů, které jsou pak do hlavního souboru `*.tex` načítány příkazem `\input`. Název pomocných souborů je odvozen z názvu hlavního `Rnw` souboru, buď to přidáním čísel, nebo se za název připojí obsah proměnné `label` (viz obr. 5).

Pro globální změnu přednastavené hodnoty parametrů, použijeme v LaTeXovském zdrojovém souboru příkaz `\SweaveOpts`, v jehož argumentu jsou uvedeny příslušné parametry a jejich hodnoty (oddělené čárkou). Např.

```
\SweaveOpts{split=TRUE, keep.source=TRUE}
```

Tyto hodnoty parametrů se použijí ve všech následujících blocích kódu pro R.

Výsledky mezi textem v odstavcovém módu

Výsledky po vyhodnocení bloku jsou vkládány do textu mezi odstavce v prostředí `Schunk`. Pro vkládání jednotlivých čísel přímo do textu v odstavci slouží příkaz `\Sexpr{`*výraz v R*`}`. Argumentem tohoto příkazu je jediný výraz jazyka R, obvykle to bývá jedna proměnná, která je zavedena na jiném místě.

```
<<results=hide>>=
x <- c(1,2,3,4)
m <- mean(x)
@
Aritmetický průměr
vektoru $x$ je \Sexpr{m}.
```

```
> x <- c(1,2,3,4)
> m <- mean(x)
```

Aritmetický průměr vektoru x je 2.5.

Uvedeno v souboru `prvni.Rnw`

```
<<split=TRUE>>=
x <- 1:6
mean(x)
@

<<sd-chunk, split=TRUE>>=
x <- 7:10
sd(x)
@
```

Byl vytvořen soubor `prvni-01.tex`

```
\begin{Schunk}
\begin{Sinput}
> x <- 1:6
> mean(x)
\end{Sinput}
\begin{Soutput}
[1] 3.5
\end{Soutput}
\end{Schunk}
```

Byl vytvořen soubor `prvni-sd-chunk.tex`

```
\begin{Schunk}
\begin{Sinput}
> x <- 7:10
> sd(x)
\end{Sinput}
\begin{Soutput}
[1] 1.290994
\end{Soutput}
\end{Schunk}
```

V souboru `prvni.tex` je vloženo

```
\input{prvni-01}

\input{prvni-sd-chunk}
```

Obrázek 5: Použití parametru `split`

Je důležité si uvědomit, že ačkoliv v R funkce `Sweave` vyhodnotí argument příkazu `\Sexpr`, funkce `Stangle` jej ignoruje. Zatímco při zpracování funkcí `Sweave` je tedy možné použít konstrukci

```
\Sexpr{(a <- 2+2)}
```

která v R do proměnné `a` vloží součet, vyhodnotí a vypíše číslo 4, přičemž proměnná `a` bude k dispozici pro další výpočty na jiném místě, stejná konstrukce zpracovaná za pomoci `Stangle` bude vypisovat chybovou hlášku o chybějící proměnné `a`.

Obrázky

Obrázky a ilustrace vytvářené programem R je do LaTeXovského souboru možné začlenit volbou parametru `fig` a `include`.

`fig` (`FALSE`/`TRUE`) — příznak pro vytváření grafického výstupu. Názvy souborů s obrázky jsou odvozeny z názvu hlavního souboru přidáním čísel, nebo obsahu proměnné `label`.

`width`, `height` (6, číslo) — rozměry obrázku v palcích.

`pdf`, `eps` (`TRUE`/`FALSE`) — příznak pro typ grafického výstupu. Uplatní se jen, pokud je nastaveno `fig=TRUE`.

Ukažme si příklad bloku, jehož výsledkem je obrázek. (Výsledný obrázek viz obr. 6.)

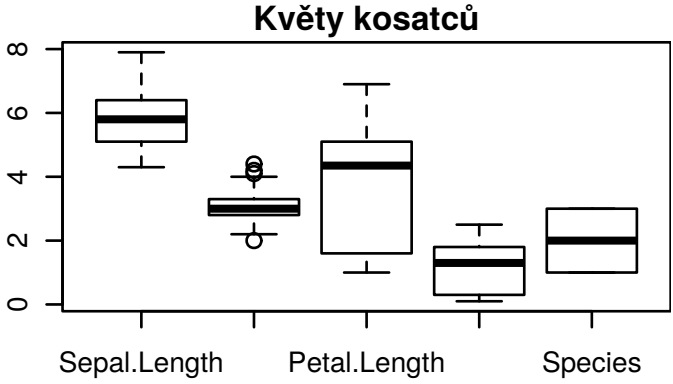
```
<<fig=TRUE, echo=FALSE, include=TRUE, width=3.5, height=2>>=
boxplot(iris, main="Květy kosatců")
@
```

Zařazení obrázků přináší dva hlavní problémy. Běžné utility programu R pro tvorbu obrázků ani do souborů PDF ani do PostScriptu nevkládají fonty, což ztěžuje přenositelnost výsledných souborů vzniklých kompilací TeXem. S tím okrajově souvisí druhý problém, nesoulad ve velikosti a typu písma v obrázku a v běžném textu. Obě tyto potíže lze překonat použitím vhodné grafické knihovny programu R. Grafickou knihovnu použitou funkcí `Sweave` nastavíme v parametru `grdevice`.

`grdevice` (text) — název funkce použité pro vytváření grafického výstupu ve `Sweave`. Je nezávislý na parametrech `pdf` a `eps`.

Jeden z dostupných grafických nástrojů v R je např. knihovna `Cairo`, jejíž použití s fonty LatinModern je v následující ukázce.

```
<<results=hide, echo=false>>=
require(Cairo)
```



Obrázek 6: Běžný grafický výstup z programu R

```
mainfont <- "lm roman 10"
CairoFonts(
    regular = paste(mainfont, "style=regular", sep=":"),
    bold = paste(mainfont, "style=bold", sep=":"),
    italic = paste(mainfont, "style=italic", sep=":"),
)

my.Swd.pdf <- function(name, width, height, ...) {
    CairoPDF(file = name,
        pointsize=10,
        width = width, height = height
        )
    }
@
```

Následně je nutné nastavit parametr `grdevice`.

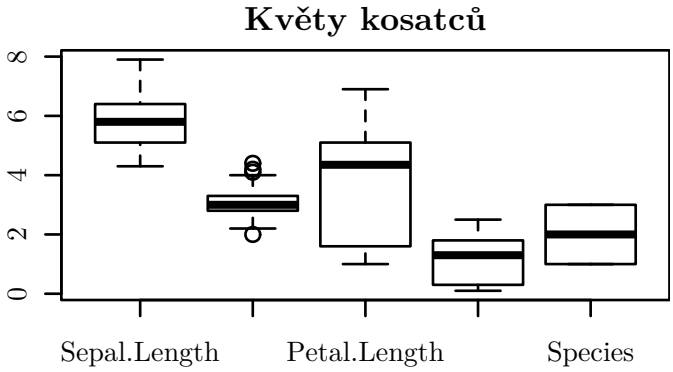
```
\SweaveOpts{grdevice=my.Swd.pdf}
```

V popiscích obrázku z předchozího příkladu jsou nyní nastaveny fonty Latin-Modern v příslušné velikosti (viz obr. 7).

Parametr `grdevice` můžeme použít i lokálně jako jeden z parametrů v jednom bloku.

V některých instalacích programu R je nejprve třeba LatinModern fonty *jednorázově* zavést do programu R.

```
> install.packages("extrafont")
> library(extrafont)
```



Obrázek 7: Obrázek s popisky ve fontu v LatinModern

```
> font_import(pattern = "lmodern*")
```

V každém následujícím použití programu R už budou LatinModern fonty k dispozici.

Obrázky jsou funkcí `Sweave` vygenerovány do pracovního adresáře a vkládány příkazem `\includegraphics` do TEXovského souboru bezprostředně na místo, kde je blok umístěn. Názvy obrázků jsou odvozeny z názvu hlavního `*.Rnw` souboru a pořadového čísla bloku, resp. z parametru `label`. Jiné umístění obrázku v textu umožní parametr `include`.

`include` (`TRUE`/`FALSE`) — vloží výstup bezprostředně na místo, kde je blok s kódem uveden, nebo v případě `FALSE` nevkládá `Sweave` do souboru `*.tex` nic a je možné vložit výstup manuálně.

Například v souboru `clanek.Rnw` uvedený blok

```
<<obr1, echo=FALSE, results=hide, include=FALSE>>=
par(mar=c(2,2,2,0))
boxplot(iris, main="Květy kosatců")
@
```

vytvoří obrázek s názvem `clanek-obr1.pdf`, ale nikam do TEXovského souboru nevkládá příkaz `\includegraphics`. To je nutné udělat ručně kdekoliv v souboru `clanek.Rnw`.

```
\begin{figure}[ht]
    \includegraphics{clanek-obr1}
    \caption{Popis obrázku}
\end{figure}
```

Použití speciálního adresáře, do kterého se ukládají obrázky, umožní parametr `prefix.string`.

`prefix.string` (text) — obsah proměnné `prefix.string` se použije při tvorbě názvů souborů při nastavení `split=TRUE` nebo `fig=TRUE` namísto názvu hlavního souboru `Rnw`.

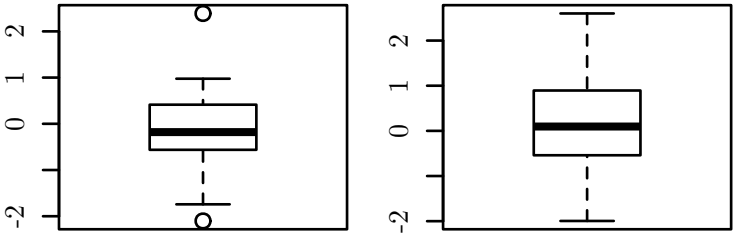
Nastavení `prefix.string=obrazky/fig` uloží všechny obrázky do adresáře `obrázky` a názvy obrázků budou složeny ze znaků `fig-` a návěští (resp. pořadového čísla) bloku. Adresář `obrazky` musí v souborovém systému existovat.[4]

Funkce `Sweave` nemá přehled o tom, jak grafická knihovna vytváří obrázky, takže není možné v jednom bloku vytvářet více obrázků najednou. Tj. konstrukce podobná následující není přípustná.

```
<<include=TRUE, fig=TRUE>>=
x <- rnorm(30)
boxplot(x)
hist(x)
@
```

Obrázky můžeme seskupit tak, že každý z obrázků vykreslíme ve zvláštním bloku s parametrem `include=FALSE` a vložíme je na správnou pozici pomocí `includegraphics` a možností LaTeXu. Skupinu obrázků lze také vytvořit s využitím vlastností programu R pomocí parametru `mfrow`

```
<<include=TRUE, fig=TRUE, width=4, height=1.5, echo=FALSE>>=
x <- rnorm(30) # generujeme náhodná data
y <- rnorm(30)
par(mfrow=c(1,2)) # obrázky zakreslíme do matice
                  # s jedním řádkem a dvěma sloupci
boxplot(x)
boxplot(y)
@
```

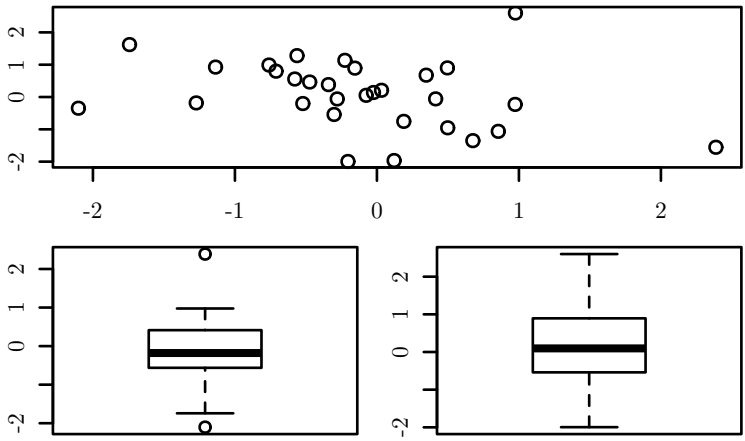


nebo funkce `layout`.

```
<<include=TRUE, fig=TRUE, width=4, height=2.5, echo=FALSE>>=
```

[4] Uživatele TeXu nepřekvapí UNIXová notace pro oddělovače vnořených adresářů.

```
layout(matrix(c(1,1,2,3), byrow=TRUE, nrow=2))
    # obrázky zakreslujeme do matice 2x2, kde první obrázek
    # vyplňuje celý první řádek a druhý a třetí obrázek jsou
    # umístěny vedle sebe na druhém řádku
plot(y ~ x)
boxplot(x)
boxplot(y)
@
```



Jako další příklad použití funkce layout by mohlo sloužit např.

```
> layout(matrix(c(1,3,2,1,4,4), byrow=FALSE, nrow=3))
```

kterému odpovídá schéma obrázků o dvou sloupcích a tří řádcích vyplněné následovně

$$\begin{pmatrix} 1 & 1 \\ 3 & 4 \\ 2 & 4 \end{pmatrix} \quad \text{tj.}$$

<table>
<tr><td colspan="2">1. obrázek</td></tr>
<tr><td>3. obrázek</td><td rowspan="2">4. obrázek</td></tr>
<tr><td>2. obrázek</td></tr>
</table>

Jak pro parametr `mfrow`, tak pro funkci `layout` se redukuje velikost fontů v obrázku. Např. pro parametr `mfrow` programu R se v seskupeních 2×2 obrázků velikost fontu násobí koeficientem 0,83, pro tři a více řádků (sloupců) se použije koeficient 0,66.

Tabulky, matice

Pěkně zformátované tabulky a matice by měly být samozřejmostí v každém dokumentu vytvořeném TEXem. Balíček `Sweave` žádným způsobem neřeší problém

vytváření tabulek a nechává tuto činnost jiným specializovaným utilitám. Přestože tento článek je věnován zejména balíku `Sweave`, je vhodné vytváření tabulek v programu R alespoň částečně nastínit.

V R je podporován výstup tabulek do TEXu několika balíky. Asi nejpoužívanější by mohl být balíček `xtable`. Funkce `xtable` objekty v R konvertuje na LATEXovské tabulky v prostředí `tabular`. Lze takto např. v R vytvořit tabulku testu ANOVA (příkaz `aov`) a pomocí `xtable` ji konvertovat.

```
<<echo=FALSE, results=tex, include=TRUE>>=
library(xtable)
xtable(aov(Petal.Width ~ Species, data=iris))
@
```

Do TEXovského souboru je vložen kód

```
% latex table generated in R 3.3.2 by xtable 1.8-2 package
% Tue Jan  3 17:17:03 2017
\begin{table}[ht]
\centering
\begin{tabular}{lrrrrr}
  \hline
 & Df & Sum Sq & Mean Sq & F value & Pr($>$F) \\
  \hline
Species & 2 & 80.41 & 40.21 & 960.01 & 0.0000 \\
  Residuals & 147 & 6.16 & 0.04 &  &  \\
   \hline
\end{tabular}
\end{table}
```

připravený pro komplilaci TEXem s výsledkem:

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Species	2	80.41	40.21	960.01	0.0000
Residuals	147	6.16	0.04		

Funkcí `xtable` lze vytvářet i jiné objekty LATEXu než tabulky, např. matice. Využijeme parametr `tabular.environment` funkce `print.xtable`. Ve funkci `xtable` musíme vymazat výpis zarovnání sloupců (parametr `align`), zakázat výpis hlaviček u sloupců a řádků, zakázat mezířádkové linky a nedovolit, aby výsledný objekt byl vnořený do plovoucího prostředí.

```
$$
<<echo=FALSE, results=tex, include=TRUE>>=
matice <- matrix(1:9, nrow=3)
Xmatice <- xtable(matice, align=rep("", ncol(matice)+1))
print(Xmatice,
```

```
      tabular.environment="pmatrix",
      include.colnames=FALSE,
      include.rownames=FALSE,
      hline.after=NULL,
      floating=FALSE)
@
$$
```

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Problematika tabulek by zcela jistě zasloužila podrobnější pojednání.

Načítání dalších souborů

Načítání dalších `*.Rnw` souborů pomocí TEXovského příkazu `\input` není možné, takto se dá načíst jen čistý TEXovský kód bez bloků. Pro načtení `noweb` souborů, které budou předloženy programu R jako preprocesoru, je nutné použít sekvenci `\SweaveInput{`*soubor*`}`. Výsledky po zpracování programem R budou všechny vloženy do jediného souboru `*.tex`. Program R samozřejmě bere v úvahu změny proměnných, které jsou provedeny podle zadání v načítaném souboru.

Pokud příkazem `\SweaveInput` načítáme soubory, které nejsou jen čistý ASCII text, je třeba nastavit typ kódování při volání funkce `Sweave`, tj. použít
`Sweave('soubor.Rnw', encoding='utf8')`
nestačí jen použít v hlavním `*.Rnw` souboru LATEXovskou sekvenci
`\usepackage[utf8]{inputenc}`

Další nastavení

Funkce `Sweave` ukládá do souboru s příponou `tex` obsah a výsledky vyhodnocení bloku (s parametry `echo=TRUE`, nebo `results=verbatim`) obklopené prostředím `Schunk`, což je jen upravené prostředí `trivlist`.

Pro výpis kódu na vstupu a výstupu používá `Sweave` prostředí `Sinput` a `Soutput` založené na balíku `fancyvrb` a prostředí `Verbatim`. Jejich úprava je tedy pro uživatele jednoduchá. Např. nastavení výpisu výsledků v tomto dokumentu je

```
\DefineVerbatimEnvironment{Soutput}{Verbatim}{%
    fontshape=sl,
    xleftmargin=0pt,
    formatcom={\color{black}},
    }
```

Pro přehlednější kód je dobré změnit chování R při výpisu rozdělených dlouhých vstupních řádků. V základním nastavení se v R pokračovací řádky na vstupu značí znakem +.

```
<<keep.source=TRUE>>=                > x <- c(1,2,
x <- c(1,2,                          +        3,4)
       3,4)
@
```

Lépe by bylo nastavit pokračovací prompt ve výstupech na (dvě) mezery. Kód bude čitelnější a v elektronické podobě lze kód bez pokračovacího promptu snadno kopírovat a spouštět v programu R.

```
<<echo=FALSE, results=hide>>=
options(continue = "  ")
@
```

a dostaneme

```
<<keep.source=TRUE>>=                > x <- c(1,2,
x <- c(1,2,                                   3,4)
       3,4)
@
```

V českém jazykovém prostředí je také vhodné používat desetinou čárku namísto desetinné tečky, čehož docílíme nastavením

```
<<echo=FALSE, results=hide>>=
options(OutDec=",")
@
```

Pro uživatele „chytrých“ programů na prohlížení výstupů, propojených s editorem tak, aby na kliknutí myši editor nastavil zdrojový soubor na požadované místo (při kompilaci `pdflatex --synctex=1`), je určen parametr

```
\SweaveOpts{concordance=TRUE}
```

který uloží informace o tom, které řádky souboru `Rnw` odpovídají za kterou oblast souboru `tex`, aby propojení bylo funkční z prohlížeče až ke zdrojovému textu `Rnw`, a ne jen k mezistupni v souboru `tex`.

Závěr

Systém `Sweave` umožňuje psát dokumenty, články nebo zprávy, obsahující výsledky výpočtů, tabulky a obrázky, které dynamicky mění podobu v závislosti na vstupních datech. Pokud autor takových dokumentů dá komunitě k dispozici zdrojový kód dokumentu a vstupní data, je obrovskou výhodou možnost opakování výpočtů a ověření správnosti výsledků. Tento přístup je užitečný při týmové práci na vzniku dokumentu, kdy každý člen týmu má k dispozici na jednom místě celý

postup od dat až k finální podobě článku. Po publikaci by pak zveřejnění tohoto dokumentu a dat mohlo zprůhlednit vznik celé vědecké práce a usnadnit revizi závěrů. V neposlední řadě má také autor archivovány všechny kroky a postupy na jediném místě a nemusí spoléhat na vlastní paměť, odkud pochází ta která hodnota, tabulka, či obrázek v textu.

Reference

Gandrud, Christopher. *Reproducible Research with R and R Studio.* 1. vyd. Boca Raton, Florida, USA : Chapman and Hall/CRC, 2013. 294 s. ISBN 1-466-57284-1.

Leisch, Friedrich. *Sweave User Manual* [on-line]. 2009. [cit. 2016-12-20]. Dostupné na: `https://www.statistik.lmu.de/~leisch/Sweave/Sweave-manual.pdf`.

Leisch, Friedrich. Dynamic generation of statistical reports using literate data analysis. In W. Härdle, B. Rönz, Eds. *Compstat 2002 – Proceedings in Computational Statistics.* Heidelberg : Physica Verlag, 2002, s. 575–580. (ISBN 3-7908-1517-9.). Dostupné z DOI: `10.1007/978-3-642-57489-4_89`.

Ramsey, Norman. Literate programming simplified. *IEEE software*, 1994, **11**(5), s. 97–105. (ISSN 0740-7459.). Dostupné z DOI: `10.1109/52.311070`.

RStudio Team. *RStudio: Integrated Development Environment for R* [on-line]. 2015. [cit. 2016-12-20]. Dostupné na: `http://www.rstudio.com/`.

Xie, Yihui. *Dynamic documents with R and knitr.* 2. vyd. Boca Raton, Florida, USA : Chapman and Hall/CRC, 2013. 294 s. ISBN 1-498-71696-2.

Poděkování recenzentům Autor by rád poděkoval oběma recenzentům za cenné podněty a připomínky, které pomohly tento text vylepšit.

Summary: Well-Documented Statistical Calculations

The paper describes the usage of the literate programming paradigm in the environment of statistical computations. In particular, the package `Sweave` for writing documents with the assistence of the statistical program R is presented.

Keywords: literate programming, reproducible research, Sweave, language R.

Marek Pomp, EkF VŠB-TU Ostrava, `marek.pomp@vsb.cz`

Sazba textu označkovaného v jazyce Markdown uvnitř TEXových dokumentů

Vít Novotný

Článek pojednává o novém makrobalíku pro formáty odvozené od plain TEXu, který umožňuje do sázených dokumentů přímo vkládat pasáže v odlehčeném značkovacím jazyce Markdown. Autor popisuje motivaci pro vznik balíku a způsob, jakým balík pracuje. Použití je ilustrováno na příkladech.

Klíčová slova: Markdown, odlehčené značkování, Lua, plain TEX, LATEX, ConTEXt, Pandoc

Úvod

TEX a přidružené programy jsou vhodným nástrojem pro sazbu mnoha druhů dokumentu, nemusí však nutně být vhodným jazykem pro přípravu jejich obsahu. Značkovací jazyky založené na SGML a XML umožňují zachytit strukturu dokumentu, aniž by hrozilo riziko vnesení chybného příkazu, který znemožní sazbu, jako je tomu u TEXu. Podstatnou výhodou je i fakt, že dokumenty lze dále publikovat a zpracovávat i mimo TEXový svět. Při přípravě hladšího materiálu lze pak využít i tzv. odlehčené značkovací jazyky. Heslem dne je zde vizuální čistota, snadný zápis a malý poměr značkování vůči textu. Jedním z odlehčených značkovacích jazyků, původně určeným pro přípravu HTML dokumentů, je Markdown (Gruber, 2013). Typickým nástrojem pro převod Markdownu (a jeho nejrůznějších dialektů) do formátů TEXu je Pandoc (MacFarlane, 2016).

Pandoc je víceúčelový nástroj, který umožňuje v první řadě převod Markdownu na obecnější jazyky (jako je LATEX, ConTEXt[1], HTML nebo XML Docbook) a do řady výstupních formátů (jako je ODF, OOXML nebo PDF). Konfigurovat parametry převodu a přidávat dodatečná metadata lze skrz parametry zadávané na příkazové řádce a skrz metabloky v jazyce YAML zapsané v záhlaví dokumentu. Při převodu na obecnější jazyky je Pandoc schopný generovat buďto pouze fragmenty, které uživatel následně vloží do těla svého dokumentu, nebo ucelené dokumenty. V druhém případě dochází k využití šablon, uvnitř kterých lze používat i jednoduchý makrojazyk; ten má přístup k metadatům zadaným na příkazové řádce a v YAML metablocích. Detailní rozbor schopností Pandocu ve vztahu k TEXu lze nalézt ve starším článku z TUGboatu (Dominici, 2014).

[1]Program Pandoc podporuje pouze převod do TEXových formátů ConTEXtu a LATEXu. Jazyk Markdown je však natolik oblíbený, že existují i programy pro jeho převod na makra alternativních makrobalíků. Takovéto podpoře se těší např. balík OPmac (Horáček, 2016).

 DOI: 10.5300/2016-1-4/78

Pandoc je bezesporu silný a užitečný nástroj při přípravě dokumentů s řadou výstupních formátů. Z pohledu uživatele, pro kterého je hlavním výstupním formátem TEX, má však i množství slabin. Při převodu například nelze jednoduše ovlivnit výstupní makra. Pro následující text v Markdownu:

```
# Úvod {#uvod}
[...]
Po zkušenostech s~nástrojem *Pandoc* jsem se rozhodl připravit
\TeX ový makrobalík, který by netrpěl neduhy zmíněnými
v~[úvodu](#uvod).
```

vygeneruje Pandoc 1.17.0.3 následující LATEXový výstup:

```
\hypertarget{uvod}{\section{Úvod}\label{uvod}}
[...]
Po zkušenostech s~nástrojem \emph{Pandoc} jsem se rozhodl
připravit \TeX ový makrobalík, který by netrpěl neduhy zmíněnými
v~\protect\hyperlink{uvod}{úvodu}.
```

Uživatel může makra `\hypertarget`, `\section`, `\label`, `\protect` a `\hyperlink` lokálně zadefinovat tak, aby jednotlivé markdownové značky odpovídaly zamýšlenému užití; nemá však garantováno, že tento výstup zůstane zachován i v budoucích verzích nástroje.

TEXové příkazy a jejich parametry jsou ve výstupním dokumentu zachovány beze změny, což autorovi umožňuje vnést do textu dokumentu chybné příkazy, které znemožní sazbu. Při použití stroje LuaTEXu bez volby `-safer`, nebo TEXového stroje s povoleným přístupem k příkazové řádce operačního systému se pak jedná i o bezpečnostní riziko. Pro detekci TEXových příkazů a jejich parametrů je použita heuristika; v textu, který není vyhodnocen jako TEXový příkaz, jsou veškeré speciální znaky plain TEXu (včetně vlnek) nahrazeny kódem, který je v příslušném TEXovém formátu vysází. Toto komplikuje vkládání TEXových příkazů, které nejsou ve formátu `\příkaz{`⟨*parametr*⟩`}`, a díky využití heuristiky není garantováno, že stejný výstup zůstane zachován i v budoucích verzích nástroje. Stejný problém sužuje i matematiku ve vstupním dokumentu.

Knuthův TEX je od roku 1989 pouze udržován. Dokumenty psané ve stabilním formátu, jako je plain TEX, a nezávislé na externích makrobalících proto při překladu TEXem dávají stejné výsledky nezávisle na verzi TEXové distribuce. Pokud uživatel využije aktivně vyvíjené TEXové formáty, makrobalíky a stroje, vyžadují již dokumenty údržbu a s novými verzemi TEXové distribuce se může měnit jejich výstup. Pokud však uživatel ve svých dokumentech nevyužívá systémové fonty ani makrobalíky spouštějící programy mimo TEXovou distribuci, měl by obdržet při překladu se stejnou verzí TEXové distribuce stejné výsledky. Pandoc je externí nástroj, který není obsažen v TEXových distribucích. Při jeho využití tedy nedává ani verze TEXové distribuce garanci stabilního výstupu překládaného dokumentu.

Dalším z nepříjemných důsledků odtržení Pandocu od TEXových distribucí je jeho absence na platformách vybudovaných nad TEXovými distribucemi. Služby jako `http://overleaf.com/` a `http://www.sharelatex.com/` umožňují spolupráci několika autorů na jednom LATEXovém dokumentu v reálném čase. Podobně jako na wiki sítích, i zde by často dávalo smysl využít okleštěný, ale snadno přístupný formát Markdownu.

Vznik makrobalíku Markdown

Po zkušenostech s nástrojem Pandoc jsem se v listopadu roku 2015 rozhodl připravit TEXový makrobalík, který by netrpěl neduhy zmíněnými v úvodu. Balík měl umožnit volně prokládat TEXový zdrojový kód textem naznačkovaným v Markdownu. Vykreslování jednotlivých markdownových značek pak mělo být řízeno TEXovými makry. Ty by sice obsahovaly rozumnou výchozí definici pro příslušný TEXový formát, ale byly by snadno předefinovatelné uživatelem.

Prvotní otázkou bylo, jakou technologii pro vývoj makrobalíku použít. V raných fázích návrhu jsem vytvořil prototyp pro plain TEX, který pomocí aktivních znaků rozpoznával redukovanou variantu jazyka Markdown. Podobné balíky již existují (Lücк, 2015), ale typicky rozpoznávají pouze jednoduché regulární a LL(k) gramatiky garantující časovou složitost $\mathcal{O}(n)$ a prostorovou složitost $\mathcal{O}(k)$. Jazyk Markdown však obsahuje ostře kontextové prvky. Existující parsery reprezentují jazyk PEG gramatikami, nebo provádí rekurzivní sestup, a na vybraných vstupech dosahují díky backtrackingu časové složitosti $\mathcal{O}(2^n)$, nebo díky memoizaci prostorové složitosti $\mathcal{O}(n)$ (Ford, 2004, sekce 6).

Alternativou bylo napsat parser odděleně v jiném programovacím jazyce, případně použít již existující parser. Inspirací mi byl LATEXový makrobalík minted (Poore, 2016), který slouží k zvýrazňování syntaxe zdrojových kódů. Minted předává zvýrazňovaný kód pythonové knihovně Pygments, která jej rozparsuje, zvýrazní a výstup v LATEXu navrátí zpět makrobalíku minted. Komunikace probíhá skrz pomocné soubory a výstupní proud 18 (příkaz `\write18`), přes který moderní TEXové stroje zpřístupňují příkazovou řádku operačního systému. Namísto Pythonu jsem však chtěl použít jazyk Lua, jehož interpret `texlua` se již nachází v TEXových distribucích. V případě použití LuaTEXu by navíc bylo možné kód spouštět přímo, aniž by bylo třeba využívat `\write18`, jehož povolení je bezpečnostním rizikem.

Začal jsem se pídit po existujících parserech Markdownu psaných v jazyce Lua. Podstatné pro mě bylo, aby parser závisel výhradně na knihovnách, které jsou staticky přilinkované k stroji LuaTEXu, aby jej bylo možné upravit a sublicencovat pod LPPL 1.3 a aby parser vyhovoval specifikaci jazyka Markdown. Jako vyhovující se ukázal balík Lunamark (MacFarlane, 2012), který je shodou okolností dílem autora nástroje Pandoc. Lunamark závisel na knihovnách LPeg, Cosmo, Selene

Unicode a Alt-getopt, z nichž klíčovými pro fungování parseru byly LPeg a Selene Unicode. Obě tyto knihovny byly staticky přilinkovány k stroji LuaTEXu (LuaTEX dev. team, 2016, sekce 3.3). Kód balíku byl zároveň uvolněn pod permisivní licencí MIT a měl k sobě připojenou sadu kvalitních regresních testů.

Lunamark sestával z 44 souborů v jazyce Lua. Jednalo se o moduly pro výstupní formáty, spustitelné soubory tvořící rozhraní pro příkazovou řádku a soubory s pomocnými definicemi. Z neaktivity na repozitářích Lunamarku a komunikace s autorem bylo zřejmé, že balík je již pouze udržován; mohl jsem tedy provést výrazné změny, aniž bych si zavíral cestu k začleňování budoucích aktualizací. Balík jsem zredukoval na jeden Lua soubor vhodný pro zanesení do TEXových distribucí. Parser bylo třeba dále upravit tak, aby ve vstupu nerozpoznával HTML kód a aby na výstupu místo materiálu k sazbě navracel syntaktický strom vstupního dokumentu představovaný TEXovými makry. Na vývoj byly vyhrazeny prostředky v rámci programu pro podporu studentských výzkumných a vývojových projektů na Fakultě informatiky Masarykovy univerzity v Brně. Kolem přepracovávaného balíku jsem začal budovat makrobalík pro TEXový formát plain TEXu a v zájmu uživatelské přívětivosti i pro formáty LATEXu a ConTEXtu ve verzích Mark II a Mark IV. Makrobalík jsem nazval Markdown. První veřejnou verzi jsem na archiv `http://ctan.org/` umístil v červnu roku 2016 (Novotný, 2016).

Architektura a použití makrobalíku Markdown

Rozhraní pro jazyk Lua

Na nejnižší úrovni je možné balík využívat z jazyka Lua. Rozhraní poskytuje metody pro konverzi textu v kódování UTF-8 a v jazyce Markdown do zdrojového textu plain TEXu a je implementováno v souboru `markdown.lua`. Pokud vytvoříme soubor `skript.lua` s následujícím obsahem:

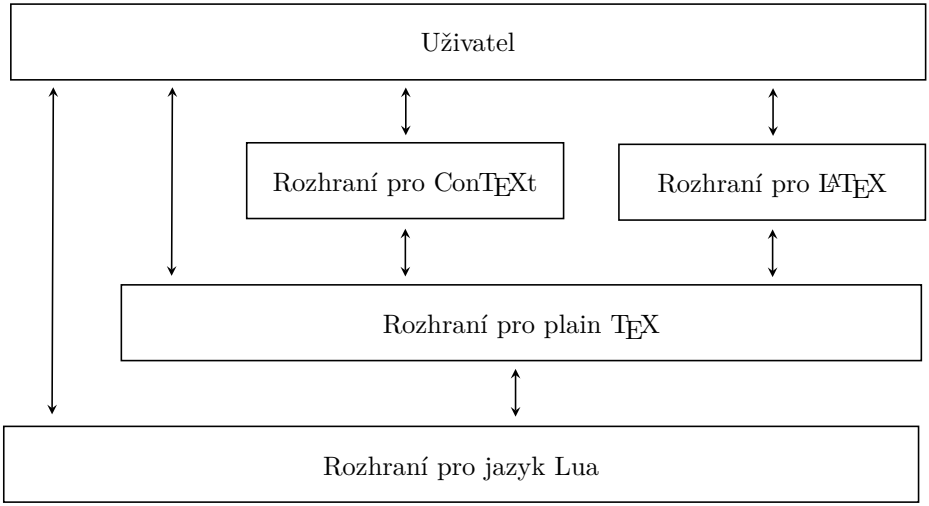
```
#/usr/bin/env texlua
local kpse = require"kpse"
kpse.set_program_name"kpsewhich"
local markdown = require"markdown"
local convert = markdown.new()
print(convert"Makrobalík jsem nazval *Markdown*.")
```

a přeložíme jej příkazem `texlua skript.lua`, měli bychom obdržet výstup v následujícím tvaru:

```
\input"./1cc0428cde58007078df6dfa2818ef75.md.tex"\relax
```

Soubor vkládaný příkazem `\input` pak obsahuje následující kód:

```
Makrobalík jsem nazval \markdownRendererEmphasis{Markdown}.\relax
```



Obrázek 1: Blokový diagram znázorňující architekturu balíku Markdown

Následující kód pak ilustruje použití rozhraní pro jazyk Lua v rámci stroje LuaTEXu při použití TEXového formátu LATEXu:

```
\documentclass{minimal}
\usepackage{fontspec}
\begin{document}
  \let\markdownRendererEmphasis=\emph
  \directlua{
    local markdown = require"markdown"
    local convert = markdown.new()
    tex.sprint(convert[[
      Makrobalík jsem nazval *Markdown*.
    ]]) }
\end{document}
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `lualatex priklad` obdržíme na výstupu dokument obsahující text: „Makrobalík jsem nazval *Markdown*.“

Metodě `markdown.new()` lze jako první argument předat tabulku obsahující parametry překladu z Markdownu. Důležitým parametrem je volba `hybrid`, která umožňuje konfigurovat, jakým způsobem se parser chová k speciálním znakům plain TEXu ve vstupním textu. Ve výchozím nastavení parser veškeré speciální znaky ve vstupu nahrazuje makry `\markdownRenderer`⟨*Značka*⟩, což je ideální

pro zpracování externích dokumentů, kterým nechceme povolit spouštět TEXové příkazy. Naopak při ruční přípravě dokumentů nám může přijít vhod hybridní režim, ve kterém parser speciální znaky plain TEXu ponechává beze změny. Následující kód v jazyce Lua:

```
#/usr/bin/env texlua
local kpse = require"kpse"
kpse.set_program_name"kpsewhich"

local markdown = require"markdown"
local convert_safe   = markdown.new()
local convert_unsafe = markdown.new({ hybrid = true })
local input  = [[
  Víme, že $1+2=3$, ale \TeX{} nám to umožní i~nádherně vysázet.
]]

print(convert_safe(input))
print(convert_unsafe(input))
```

nám dává při překladu interpretem `texlua` na výstupu dvojici dokumentů. V prvním jsou veškeré výskyty speciálních znaků plain TEXu nahrazeny makry:

```
Víme, že \markdownRendererDollarSign{}1+2=3%
\markdownRendererDollarSign{}, ale \markdownRendererBackslash{}%
TeX\markdownRendererLeftBrace{}\markdownRendererRightBrace{}
nám to umožní i\markdownRendererTilde{}nádherně vysázet.\relax
```

Druhý je pak až na závěrečný `\relax` totožný se vstupem.

Nabídka značek v základním Markdownu je značně omezená. Skrze parametry překladu lze proto aktivovat i různorodá rozšíření syntaxe. V době přípravy článku balík podporuje rozšíření pro citace, poznámky pod čarou, zdrojové kódy s vyznačením názvu programovacího jazyka a definiční seznamy:

```
#/usr/bin/env texlua
local kpse = require"kpse"
kpse.set_program_name"kpsewhich"

local markdown = require"markdown"
local convert = markdown.new({
  citations       = true, -- Rozšíření pro citace
  footnotes       = true, -- Rozšíření pro poznámky pod čarou
  fencedCode      = true, -- Rozšíření pro zdrojové kódy
  definitionLists = true, -- Rozšíření pro definiční seznamy
})
```

```
local input  = [[
  @doe09 tvrdí, že časová složitost následujícího řadicího
  algoritmu je lineárně logaritmická[^sleepsort]:

  ``` sh
  #!/bin/sh
  for N; do
    (sleep $N; echo $N) &
  done
  wait
  ``````

  [^sleepsort]: Kvadratická, pokud uvážíme práci plánovače jádra.

  žába
  :   slizká
  :   zelená [viz -@smith12, s. 123]
  :   kvákající
]]

print(convert(input))
```

Při překladu interpretem `texlua` pak obdržíme následující dokument:

```
\markdownRendererTextCite{1}+{}{}{doe09} tvrdí, že časová
složitost následujícího řadicího algoritmu je lineárně
logaritmická\markdownRendererFootnote{Kvadratická, pokud uvážíme
práci plánovače jádra.}:\markdownRendererInterblockSeparator
{}\markdownRendererInputFencedCode{./04602c8f5c223967cfe7aa8282d%
6f82f.verbatim}{sh}\markdownRendererInterblockSeparator
{}\markdownRendererDlBeginTight\markdownRendererDlItem{žába}%
\markdownRendererDlDefinitionBegin slizká%
\markdownRendererDlDefinitionEnd
\markdownRendererDlDefinitionBegin zelená \markdownRendererCite
{1}-{viz}{s.\markdownRendererNbsp{}123}{smith12}%
\markdownRendererDlDefinitionEnd
\markdownRendererDlDefinitionBegin kvákající%
\markdownRendererDlDefinitionEnd\markdownRendererDlItemEnd
\markdownRendererDlEndTight\relax
```

Výčet známých parametrů nalezneme v dokumentaci (Novotný, 2016, sekce 2.1.2).

Rozhraní pro plain TEX

Pokud zamýšlíme výstup parseru bezprostředně sázet, můžeme s výhodou využít plainTEXové rozhraní, které nás odstíní od komunikace s interpretem jazyka Lua. To oceníme především při práci s TEXovými stroji, které skriptování v jazyce Lua přímo neumožňují (pdfTEX, XƎTEX). PlainTEXové rozhraní je implementováno v souboru `markdown.tex`, který zavedeme pomocí příkazu `\input markdown`.

Rozhraní definuje makra `\markdownRenderer`⟨*Značka*⟩, která se mohou nacházet ve výstupu metod luového rozhraní. Makra `\markdownRenderer`⟨*Značka*⟩ implicitně expandují na makra `\markdownRenderer`⟨*Značka*⟩`Prototype`, která pak obsahují výchozí definici pro jednotlivé markdownové značky. Toto rozdělení má následující význam: makra `\markdownRenderer`⟨*Značka*⟩ jsou určena k předefinování uživatelem, zatímco makra `\markdownRenderer`⟨*Značka*⟩`Prototype` jsou určena pro tvůrce makrobalíků. Ti jim mohou nastavit výchozí hodnoty, které považují za rozumné, aniž by mohlo dojít k přepsání uživatelovy konfigurace:

```
% Tento makrobalík slouží pro sazbu kuchařských receptů.
\def\nadpis#1{Recept: #1}
% [...]
\ifx\markdownVersion\undefined\else
  \let\markdownRendererHeadingOnePrototype=\nadpis
\fi
```

Úplný výčet značek lze nalézt v dokumentaci (Novotný, 2016, sekce 2.2.3).

PlainTEXové rozhraní rovněž poskytuje přístup k parametrům překladu z luového rozhraní. Konkrétně platí, že pro libovolný ⟨*parametr*⟩ je dostupné makro `\markdownOption`⟨*Parametr*⟩, které může uživatel předefinovat a ovlivnit překlad:

```
\def\markdownOptionHybrid{true}%
\def\markdownOptionFencedCode{true}%
```

Kromě parametrů překladu umožňuje plainTEXové rozhraní nastavovat i další parametry, které souvisí s komunikací s interpretem jazyka Lua. Úplný výčet rozpoznávaných maker lze nalézt v dokumentaci (Novotný, 2016, sekce 2.2).

Text v Markdownu lze zapsat mezi příkazy `\markdownBegin` a `\markdownEnd`:

```
\input markdown
\def\markdownRendererEmphasis#1{\it#1}%
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\bye
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `pdfcsplain -shell-escape priklad` obdržíme na výstupu dokument obsahující text „Makrobalík jsem nazval *Markdown*.“

S použitím příkazů `\markdownBegin` a `\markdownEnd` se pojí několik nedostatků. Markdown například umožňuje do výstupu vložit řádkový zlom tak, že uživatel na konec řádku ve vstupu přidá dvě a více mezer. V TEXu jsou však veškeré mezery na konci řádku zahozeny ještě před tím, než se text objeví ve vstupním bufferu (Knuth, 1986, strana 46). Při použití příkazů `\markdownBegin` a `\markdownEnd` je vstupní text v Markdownu načítán TEXem a řádkové zlomy tedy není možné detekovat. Další drobné nedostatky příkazů `\markdownBegin` a `\markdownEnd` jsou popsány v dokumentaci (Novotný, 2016, sekce 2.2.1).

Pokud máme markdownový text uložený v externím souboru, můžeme jej do TEXového dokumentu vložit pomocí příkazu `\markdownInput`:

```
\input markdown
\markdownInput{priklad.md}%
\bye
```

Nedostatky spojené s použitím příkazů `\markdownBegin` a `\markdownEnd` se na příkaz `\markdownInput` nevztahují.

Rozhraní pro LaTeX

TEXový formát LaTeXu implementuje většinu maker plain TEXu (Braams et al., 2016, sekce 9). V rámci LaTeXu však nelze používat přímo plainTEXové rozhraní:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\input markdown
\begin{document}
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\end{document}
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `lualatex priklad` obdržíme následující chybu:

```
Module luatexbase Error: Attempt to use callback.register()
(luatexbase)             directly on input line 6
```

PlainTEXová implementace pro LuaTEX využívá při načítání vstupního textu háček `process_input_buffer` (LuaTEX dev. team, 2016, sekce 8.3.1). LaTeX však pro načítání textu poskytuje vlastní rozhraní (Braams et al., 2016, sekce 73.4).

Pokud dokument přeložíme příkazem `pdflatex -shell-escape priklad`, obdržíme následující chybu:

```
! Package inputenc Error: Unicode char ?kj (U+E2)
(inputenc)                not set up for use with LaTeX.
```

```
l.1 Makrobal?k j
                sem nazval \markdownRendererEmphasis{Markdown}...
```

PlainTEXová implementace pro stroj pdfTEXu je schopna si poradit se speciálními znaky plain TEXu ve vstupu, ale nepočítá s použitím LATEXového balíku inputenc, který počáteční bajty znaků mimo ASCII učiní TEXovými aktivními znaky.

Z těchto důvodů je vhodné použít přímo LATEXové rozhraní. To je implementováno v souboru `markdown.sty`, který zavedeme pomocí příkazu `\usepackage[`⟨*parametry*⟩`]{markdown}`. Jak znázorňuje obrázek 1 ze str. 82, zavádí LATEXová implementace zároveň plainTEXovou implementaci. Můžeme tedy přímo používat makra z plainTEXového rozhraní:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{markdown}
\let\markdownRendererEmphasis=\emph
\begin{document}
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\end{document}
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazy `lualatex priklad` nebo `pdflatex -shell-escape priklad` obdržíme na výstupu dokument obsahující text: „Makrobalík jsem nazval *Markdown*.“

Nad úroveň plainTEXového rozhraní zavádí LATEXové rozhraní navíc příkaz `\markdownSetup{`⟨*parametry*⟩`}`, LATEXová prostředí `\begin{markdown}`...`\end{markdown}` a `\begin{markdown*}{`⟨*parametry*⟩`}`...`\end{markdown*}` a zakrývá plainTEXový příkaz `\markdownInput{`⟨*vstupní soubor*⟩`}` svým příkazem `\markdownInput[`⟨*parametry*⟩`]{`⟨*vstupní soubor*⟩`}`.

Pomocí příkazů `\usepackage[`⟨*parametry*⟩`]{markdown}` a `\markdownSetup{`⟨*parametry*⟩`}` můžeme nastavovat parametry překladu z luového rozhraní:

```
\usepackage[
  citations,        %% Rozšíření pro citace
  footnotes,        %% Rozšíření pro poznámky pod čarou
]{markdown}
\markdownSetup{
  fencedCode,       %% Rozšíření pro zdrojové kódy
  definitionLists, %% Rozšíření pro definiční seznamy
}
```

Stejně tak můžeme pomocí parametrů `renderers` a `rendererPrototypes` nastavovat makra z plainTEXového rozhraní ve tvaru `\markdownRenderer`⟨*Značka*⟩ a `\markdownRenderer`⟨*Značka*⟩`Prototype`:

```
\markdownSetup{
  renderers = {
    emphasis = {\emph{#1}},
  }, rendererPrototypes = {
    link = {\href{#1}{#3}},
  }
}
```

Vzhledem ke způsobu, jakým LaTeX 2$_\varepsilon$ čte parametry příkazu `\usepackage[`⟨*parametry*⟩`]{markdown}`, nemohou ⟨*parametry*⟩ obsahovat víceodstavcový text ani argumenty maker (například `#1`). V praxi toto činí parametry `renderers` a `rendererPrototypes` nepoužitelnými, a proto jsou v rámci příkazu `\usepackage` zakázány.

LaTeXové prostředí `\begin{markdown}`...`\end{markdown}` je svou funkcí totožné s příkazy `\markdownBegin` a `\markdownEnd` z plain TEXového rozhraní. Zajímavějšími pro nás budou LaTeXové prostředí `\begin{markdown*}{`⟨*parametry*⟩`}`...`\end{markdown*}` a příkaz `\markdownInput[`⟨*parametry*⟩`]{`⟨*vstupní soubor*⟩`}`, které nám podobně jako příkaz `\markdownSetup{`⟨*parametry*⟩`}` umožňují nastavovat parametry překladu z luového rozhraní a makra `\markdownRenderer`⟨*Značka*⟩ a `\markdownRenderer`⟨*Značka*⟩`Prototype` z plainTEXového rozhraní:

```
\documentclass{article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{markdown}
\markdownSetup{renderers = {
  emphasis = {\textbf{#1}},
}}
\begin{document}
\begin{markdown*}{renderers = {
  emphasis = {\emph{#1}}
}}
  Makrobalík jsem nazval *Markdown*.
\end{markdown*}
\begin{markdown*}{hybrid}
  Víme, že $1+2=3$, ale \TeX{} nám to
  umožní i~nádherně *vysázet*.
\end{markdown*}
\end{document}
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `pdflatex -shell-escape priklad` obdržíme na výstupu dokument obsahující text „Makrobalík jsem nazval *Markdown*. Víme, že $1 + 2 = 3$, ale TEX nám to umožní i nádherně **vysázet**.“ Vidíme, že parametry LATEXového prostředí `\begin{markdown*}{`⟨*parametry*⟩`}`...`\end{markdown*}` jsou zpracovány způsobem, který odpovídá naší intuici, a mají pouze lokální platnost. Příkaz `\markdownInput[`⟨*parametry*⟩`]{`⟨*vstupní soubor*⟩`}` se chová analogicky.

LATEXové rozhraní se snaží nastavit rozumné výchozí hodnoty pro makra `\markdownRenderer`⟨*Značka*⟩`Prototype` z plainTEXového rozhraní. Při běžném použití proto můžeme konfiguraci balíku přeskočit a rovnou se pustit do psaní. Výchozí hodnoty maker jsou popsány v dokumentaci (Novotný, 2016, sekce 3.3.4).

Rozhraní pro ConTEXt

TEXové formáty ConTEXt Mark II a Mark IV implementují většinu maker plain TEXu. Podobně jako u LATEXu však nelze používat přímo plainTEXové rozhraní:

```
\input markdown
\starttext
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\stoptext
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `texexec --passon=-shell-escape priklad` obdržíme následující chybu:

```
! Undefined control sequence.
\markdownReadAndConvert ...de `##1=12}\dospecials
                                                  \catcode `\ ...
l.3 \markdownBegin
```

TEXový formát ConTEXt Mark II nedefinuje plainTEXový příkaz `\dospecials`. Pokud definici doplníme:

```
\input markdown
\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\_\do\%\do\~}%
\starttext
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\stoptext
```

a překlad opakujeme, podaří se nám již překlad dokončit. Pokud však dokument přeložíme příkazem `context priklad`, obdržíme v terminálu následující varování:

```
system> callbacks > not registering frozen 'process_input_buffer'
```

a překlad nikdy neskončí. PlainTEXová implementace využívá při načítání vstupního textu v stroji LuaTEXu háček `process_input_buffer` (LuaTEX dev. team, 2016, sekce 8.3.1). Podobně jako LATEX však formát ConTEXt Mark IV háčky LuaTEXu využívá interně a pro načítání textu poskytuje vlastní rozhraní.

Z těchto důvodů je vhodné použít přímo ConTEXtové rozhraní. To je implementováno v souboru `t-markdown.tex`, který zavedeme příkazem `\usemodule [t][markdown]`. Jak znázorňuje obrázek 1 ze str. 82, zavádí ConTEXtová implementace zároveň plainTEXovou implementaci. Můžeme tedy přímo používat makra z plainTEXového rozhraní:

```
\usemodule[t][markdown]
\let\markdownRendererEmphasis=\emph
\starttext
\markdownBegin
  Makrobalík jsem nazval *Markdown*.
\markdownEnd
\stoptext
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `texexec --passon=-shell-escape priklad` nebo `context priklad` obdržíme na výstupu dokument obsahující text: „Makrobalík jsem nazval *Markdown*.“

Nad úroveň plainTEXového rozhraní zavádí rozhraní pro ConTEXt příkazy `\startmarkdown` a `\stopmarkdown`, které jsou svou funkcí totožné s příkazy `\markdownBegin` a `\markdownEnd` z plainTEXového rozhraní. Kromě těchto maker není pro ConTEXt definováno žádné další rozšiřující rozhraní.

Podobně jako LATEXové rozhraní se i rozhraní pro ConTEXt snaží nastavit rozumné výchozí hodnoty pro makra `\markdownRenderer`⟨*Značka*⟩`Prototype` z plainTEXového rozhraní. Při běžném použití proto můžeme konfiguraci balíku přeskočit a rovnou se pustit do psaní. Výchozí hodnoty maker jsou popsány v dokumentaci (Novotný, 2016, sekce 3.4.3).

Specifika československé sazby

V češtině a slovenštině je typografickou chybou ponechat neslabičné předložky (k, s, v, z) na koncích řádků. V TEXu tento problém řešíme ručním vložením nezlomitelné mezery mezi předložku a následující slovo, použitím programu vlna (Olšák, 2010), nebo použitím LATEXových makrobalíků XƎVlna (Wagner, 2013) a encxvlna (Olšák; Wagner, 2014).

Pokud pro překlad dokumentu využívajícího makrobalík Markdown použijeme stroj XeTeX, nebo encTeX, stačí nám zavést makrobalíky XeVlna, nebo encxvlna a nezlomitelné mezery budou automaticky doplněny na příslušná místa. Stejně tak můžeme, pokud makrobalík Markdown používáme v hybridním režimu, vložit do dokumentu nezlomitelné mezery ručně nebo pomocí programu vlna.

Pokud však makrobalík Markdown používáme mimo hybridní režim a zároveň pro sazbu nevyužíváme stroje XeTeX a encTeX, stává se vkládání nezlomitelných mezer mírně komplikovanějším. Při překladu z Markdownu se totiž veškeré znaky vlnky (v plain TeXu představující nezlomitelné mezery) na vstupu přeloží na makro `\markdownRendererTilde`, které ve výchozím nastavení znak vlnky vysází. Nejidiomatičtějším řešením je změnit definici makra `\markdownRendererTilde` tak, aby expandovalo na nezlomitelnou mezeru:

```
\input markdown
\let\markdownRendererTilde=~
\markdownBegin
  Nyní v~textu mohu zadávat nezlomitelné mezery.
\markdownEnd
\bye
```

Pokud dokument uložíme do souboru `priklad.tex`, pak při překladu příkazem `pdfcsplain -shell-escape priklad` obdržíme na výstupu dokument obsahující text „Nyní v textu mohu zadávat nezlomitelné mezery.“ Nevýhodou tohoto řešení je, že autorovi dokumentu upíráme možnost vysázet znak vlnky.

Pokud nás to trápí, můžeme se uchýlit k alternativnímu řešení. Makrobalík Markdown si přeložené soubory odkládá do adresáře zadaného makrem `\markdownOptionCacheDir` (Novotný, 2016, sekce 2.2.2.1), které implicitně expanduje na `_markdown_\jobname`. Toho využijeme v následujícím dokumentu:

```
\input markdown
\markdownBegin
  Nyní v textu mohu používat vlnky: ~.
\markdownEnd
\bye
```

Pokud dokument uložíme do souboru `priklad.tex` a přeložíme jej příkazem `pdfcsplain -shell-escape priklad`, pak nám v adresáři `_markdown_priklad` vznikne soubor s názvem ve tvaru ⟨*haš vstupu*⟩`.md.tex` a s následujícím obsahem:

```
Nyní v textu mohu používat vlnky: \markdownRendererTilde{}.\relax
```

Pokud tento soubor zpracujeme programem vlna a opět dokument přeložíme příkazem `pdfcsplain -shell-escape priklad`, pak makrobalík Markdown použije náš upravený soubor `_markdown_priklad/`⟨*haš vstupu*⟩`.md.tex` a na výstupu obdržíme dokument obsahující text: „Nyní v textu mohu používat vlnky: ~.“

Reference

Johannes Braams, David Carliste, Alan Jeffrey a kol. *The LaTeX 2ε Sources* [on-line]. 2016. [cit. 2016-06-02]. Dostupné na: `http://mirrors.ctan.org/macros/latex/base/source2e.pdf`.

Dominici, Massmiliano. An overview of Pandoc [on-line]. *TUGboat*, 2014, **35**(1), s. 44–50. [cit. 2016-08-15]. (ISSN 0896-3207.) Dostupné na: `http://tug.org/TUGboat/tb35-1/tb109dominici.pdf`.

Ford, Brian. Parsing expression grammars: A recognition-based syntactic foundation. *ACM SIGPLAN Notices*, New York, NY, USA : ACM, 2014, **39**(1), s. 111–122. (ISSN 0362-1340.). Dostupné z DOI: `10.1145/982962.964011`.

Gruber, John. *Markdown* [on-line]. 2013. [cit. 2016-08-15]. Dostupné na: `http://daringfireball.net/projects/markdown/`.

Horáček, Michal. *Markdown to OPmac converter* [on-line]. *Ver. 2dd262d 2016-06-24*. 2016. Dostupné na: `https://bitbucket.org/horacmi/md2opmac`.

Knuth, Donald Ervin. *The TeXbook*. 3. vyd. Boston : Addison-Westley, 1986. ix + 479 s. ISBN 0-201-13447-0.

LuaTeX dev. team. *LuaTeX reference manual* [on-line]. 2016. [cit. 2016-12-23]. 222 s. Dostupné na: `http://www.luatex.org/svn/trunk/manual/luatex.pdf`.

Lück, Uwe. *nicetext: Minimal markup for simple text (Wikipedia style) and documentation* [on-line]. *Ver. r0.67*. 2015. Dostupné na: `https://www.ctan.org/pkg/nicetext`.

MacFarlane, John. *lunamark* [on-line]. *Ver. 0.4.0*. 2012. Dostupné na: `http://jgm.github.io/lunamark`.

MacFarlane, John. *Pandoc: A universal document converter* [on-line]. *Ver. 1.17.2*. 2016. Dostupné na: `http://pandoc.org`.

Novotný, Vít. *A Markdown Interpreter for TeX* [on-line]. 2016. [cit. 2016-08-17]. Dostupné na: `http://mirrors.ctan.org/macros/generic/markdown/markdown.pdf`.

Olšák, Petr. *Program vlna* [on-line]. *Ver. 1.5*. 2010. Dostupné na: `http://petr.olsak.net/ftp/olsak/vlna/vlna-1.5.tar.gz`.

Wagner, Zdeněk, Olšák, Petr. *encxvlna: Vlna implemented in encTeX* [on-line]. *Ver. 1.1*. 2014. Dostupné na: `https://www.ctan.org/pkg/encxvlna`.

Poore, Geoffrey M. *The minted package* [on-line]. *Ver. 2.4*. 2016. Dostupné na: `http://ctan.org/pkg/minted`.

Wagner, Zdeněk. *XeVlna: Vlna implemented in XeTeX* [on-line]. *Ver. 1.0*. 2013. Dostupné na: `https://www.ctan.org/pkg/xevlna`.

Summary: Rendering Markdown inside TEX Documents

The article describes a new package for plain TEX derivatives that enables the direct inclusion of Markdown-formatted text into TEX documents. The author describes their motivation for the creation of the package and its inner workings. The usage of the package is explained through example.

Keywords: Markdown, lightweight markup, Lua, plain TEX, LATEX, ConTEXt, Pandoc

Vít Novotný, witiko@mail.muni.cz

Elektronické knihy a systém TEX4ebook

Michal Hoftich

Článek pojednává o konverzi dokumentů vytvořených v TEXu do formátů elektronických knih, především Epub 3. Na konkrétních příkladech jsou ukázány možnosti uzpůsobení průběhu konverze a vzhledu výsledného e-booku.

Klíčová slova: e-knihy, TeX4ht, HTML, konverze.

Elektronické knihy

Elektronické knihy (e-booky) se rozvíjejí především od 70. let 20. století, kdy byl spuštěn projekt Guttenberg, jehož cílem je digitalizace knih a textů, na které se nevztahují autorská práva. Prudký rozvoj nastal od 90. let, kdy se objevily osobní digitální asistenty (PDA) a masověji se rozšířily osobní počítače. Zpočátku je tvořili především nadšenci, komerční vydavatelství pouze pomalu ztrácela nedůvěru. V roce 2004 se objevila první čtečka s displejem s elektronickým inkoustem (e-ink), o tři roky později internetový obchod Amazon představil svojí čtečku Kindle a nastal velký komerční rozvoj.

Hlavními vlastnostmi e-booků jsou flexibilita zobrazení, neboť je třeba, aby šly zobrazit na řadě zařízení s různou velikostí a vlastnostmi displeje, možnost upravit vlastnosti zobrazení, jako je velikost nebo barva písma podle potřeb konkrétního čtenáře. Některé čtečky také umožňují předčítání textu technologií text to speech.

V průběhu vývoje se objevilo množství formátů e-booků, v dnešní době jsou nejrozšířenějšími Epub, z něho vycházející Epub 3 a Mobi, používaný Amazonem. Tyto formáty v podstatě vychází z formátů HTML a CSS, běžně používaných na WWW stránkách, a pomocných metadat. Formát Mobi se vytváří kompilací Epubu programem Kindlegen, který Amazon poskytuje pro Windows, Mac OS i Linux.

Výhodou Epubu je jeho široká podpora nejrůznější škálou zařízení a programů, které ovšem často podporují pouze některé jeho vlastnosti, často mají problém například s CSS styly, takže formátování e-booku může být zcela rozdílné, než autor zamýšlel. Nejlepší využití Epubu je pro jednoduchý text bez složitějšího formátování, například pro beletrii nebo literaturu faktu.

Totéž platí v podstatě i o formátu Mobi, který vzniká konverzí z Epub. Vlastně se jedná o dva formáty v jednom, neboť Kindlegen vytváří verzi pro novější i starší verzi svých zařízení. Amazon pro autory připravil dokument *Kindle design*

 doi: 10.5300/2016-1-4/94

guidelines, ve kterém konkrétně specifikuje, jakým způsobem by měl být zdrojový Epub soubor připraven, aby konverze proběhla nejlepším způsobem.

Protože formát Epub vznikal živelně a jeho implementace ve čtecích zařízeních je nejednotná, organizace IDPF (IDPF, 2016) vytvořila standard Epub 3. Ten by měl mimo jiné zlepšit podporu pro technické publikace nebo pro asistenční služby pro čtenáře se specifickými potřebami. Tento formát bude tedy pro uživatele TEXu patrně nejzajímavější, ovšem je třeba zmínit, že přestože je tento formát standardizovaný, čtecí zařízení zdaleka nepodporují všechny jeho vlastnosti. V současné době vzniká jeho nová verze, Epub 3.1, která obsahuje opět poměrně zásadní změny, ovšem zatím nebyla představena finální verze, proto se jím nebudeme dále zabývat.

Užitečnými vlastnostmi Epub 3 pro technické publikace jsou podpora MathML pro sazbu matematiky,[1] podpora SVG pro vektorové obrázky, podpora JavaScriptu, která umožňuje tvorbu interaktivních prvků, vkládání fontů do dokumentu. Významnou změnou jsou nové HTML elementy pro značkování logické struktury dokumentu (například poznámky pod čarou, bibliografie, rejstřík). Formát SMIL umožňuje vkládání multimediálního obsahu.

Patrně nejzajímavější ukázkou možností Epub 3, která v současnosti existuje, je projekt Aeneas (Petarin, 2016), který umožňuje synchronizovat čtenou verzi knihy s jejím textem. Takto připravené e-booky lze číst v aplikaci Menestrello (Petarin, 2016), která zvýrazňuje právě předčítaný text. Je to vynikající pomůcka například pro procvičování výuky jazyků.

Konverze TEXu na e-book

Kvůli stále rozšířenějšímu čtení textů z elektronických zařízení, ať již jsou to PC, mobilní telefony, tablety nebo e-ink čtečky, vzniká potřeba konverze z TEXu do e-booků. Všechny zmíněné formáty jsou v podstatě balíčky WWW stránek s přidanými metadaty, takže tato úloha se v podstatě skládá z konverze z TEXu do HTML, přidání nezbytných metadat a zabalení výsledných souborů do e-bookového formátu.

Pro konverzi TEXu do HTML se dá použít řada nástrojů, nejrozšířenějšími jsou LaTeX2HTML (Drakos, 2016), LaTeXML (Miller, 2016), Pandoc (MacFarlane, 2016) a TEX4ht (Gurari, 1997). První tři z nich jsou založené na zpracování TEXového dokumentu externím programem, který se snaží rozpoznat použitá makra a převést je na HTML. LaTeXML je z nich nejpokročilejší, neboť emuluje TEXovou expanzi maker, a podporuje tudíž i uživatelská makra a balíčky. Zbylé dva systémy podporují především základní příkazy LATEXu

[1] V ostatních formátech je třeba používat obrázky, což je především pro matematiku vloženou v textu nevhodné řešení.

a se složitějšími dokumenty mohou mít problém. Všechny podporují pouze LaTeX, jiné formáty nikoli.

TeX4ht je sada maker, která se automaticky načítá do dokumentu a upravuje definici jednotlivých příkazů tak, aby bylo možné vkládat formátovací značky podporovaných výstupních formátů (kromě HTML je to především OpenDocument Format, TEI nebo Docbook). Díky tomu, že pro překlad je používaný samotný TeX, neměl by být problém s užíváním uživatelských maker a balíčků. Kromě LaTeXu podporuje i jiné formáty, například Plain, ovšem u nich může být třeba upravit dokument. Pro překlad je možné použít současné nejpoužívanější TeXové enginy, tedy pdfTeX, XeTeX a LuaTeX. Systém vytvářel Eitan Gurari, po jeho smrti vývoj převzal Karl Berry, CV Radhakrishnan a autor článku. Právě TeX4ht je použit jako základ pro TeX4ebook (Hoftich, 2016a), systém pro konverzi TeXu na e-booky.

TeX4ebook

TeX4ebook je systém pro konverzi TeXu do formátů Epub, Epub 3 a Mobi, hlavní důraz je kladený na Epub 3. Hlavní výhodou oproti pouhé konverzi TeXu do HTML a jeho následné konverzi na e-book externími nástroji je možnost získání různých metadat přímo ze zpracovávaného dokumentu. Systém emuluje základní postup kompilace používaný TeX4ht a přidává několik kroků navíc. Celý postup je konfigurovatelný s použitím sestavovacího skriptu.

Základní postup je shodný s TeX4ht a sestává ze tří kroků:

1. Kompilace dokumentu TeXem. Ještě před načtením dokumentu je nahrán balíček `tex4ht.sty`, který řídí další postup zpracování dokumentu. Pro každý načtený balíček se testuje existence souboru s příponou `.4ht`, který, pokud existuje, je načten po skončení preamble dokumentu. V tomto konfiguračním souboru se mohou předefinovávat makra a vkládají se do nich takzvané *háčky*. Po nahrání všech `.4ht` souborů pro použité balíčky jsou nahrány soubory s konfigurací pro daný výstupní formát. Obsahují instrukce, které jsou vloženy do háčků definovaných v předešlém kroku. Háčky je také možné konfigurovat pomocí konfiguračního souboru, který bude popsaný dále.

   Kompilace je spouštěna několikrát, protože jsou využívány pomocné soubory pro tvorbu hypertextových odkazů. Obyčejně jsou třeba tři kompilace, aby všechny odkazy fungovaly správně, ve výjimečných případech může být třeba i více kompilací.
2. Zpracování DVI souboru příkazem `tex4ht`. Tento příkaz vytváří výstupní soubory, převádí různá vstupní kódování na `UTF-8` a vytváří dva pomocné soubory: `.idv` soubor je speciální soubor ve formátu DVI, který obsahuje stránky, které mají být převedeny na obrázky. Jedná se například o prostředí

`picture` LaTeXu, nebo matematiku, pokud není převáděna do formátu MathML. soubor `.lg` obsahuje seznam výstupních souborů, CSS instrukce a instrukce pro kompilaci jednotlivých stránek v `.idv` souboru na obrázky.

3. Zpracování `.lg` souboru příkazem `t4ht`. V tomto kroku je vytvořen CSS soubor, dochází ke konverzi obrázků a výstupní soubory mohou být předány externím programům k dalšímu zpracování.

TeX4ht používá pro kompilaci řadu skriptů, které se liší použitým TeXovým formátem, enginem a výstupním formátem. Všechny provádí tři kompilační kroky popsané výše a předávají jednotlivým příkazům argumenty v uvozovkách. Například pro vytvoření dokumentu ve formátu XHTML a kódování `UTF-8` je možné použít následující příkaz:

```
htlatex filename.tex "xhtml,charset=utf-8" " -utf8 -cunihtf"
```

Tento postup není úplně uživatelsky přívětivý, proto TeX4ebook umožňuje použít pro základní konfiguraci přepínače, známé z běžných příkazů užívaných například v Linuxu.

```
tex4ebook -f epub3 filename.tex
```

Tento příkaz vytvoří soubor `filename.epub` ve formátu Epub 3, kódování `UTF-8` je použito automaticky.

Hlavní rozdíl TeX4ebook oproti TeX4ht spočívá ve třetím kompilačním kroku. Příkaz `t4ht` je použit pouze pro vytvoření CSS souboru, konverzi obrázků a spouštění externích příkazů řídí TeX4ebook sám. Kromě toho je možné spouštět příkazy mezi jednotlivými kompilacemi TeXu, například po první kompilaci a tvorbě pomocných souborů je možné spustit příkazy pro tvorbu rejstříku a bibliografie.

Postup kompilace lze řídit pomocí sestavovacích skriptů Make4ht (Hoftich, 2016b). To je sestavovací program pro TeX4ht, který původně vznikal jako součást TeX4ebook, ale postupně se vyvinul do samostatné aplikace a knihovny, která nahrazuje skripty používané pro kompilaci TeX4ht.

Sestavovací skripty jsou programy v jazyce Lua:

```
local filter = require "make4ht-filter"
local process = filter{"hruletohr"}
Make:add("biber","biber ${input}")
Make:htlatex{}
Make:biber {}
Make:htlatex {}
Make:image("png$",
"dvipng -bg Transparent -T tight -o ${output}"..
"-pp ${page} ${source}")
Make:match("html$",process)
Make:match("html$",
"tidy -m -utf8 -asxhtml -q -i ${filename}")
```

Pomocí příkazu `Make:add("biber", "biber ${input}")` definujeme příkaz `Make:biber`, který můžeme použít pro tvorbu bibliografie s balíčkem Biblatex. Druhý argument příkazu `Make:add` je šablona s příkazem, který se spustí. Proměnné definované Make4ht se vkládají pomocí konstruktu `${jméno}`, proměnná `input` obsahuje název zpracovávaného souboru.

Argumenty ve složených závorkách pro příkazy `Make:htlatex` a `Make:biber` jsou tabulky, ve kterých můžete nastavit hodnoty proměnných užívaných v šablonách, pokud byste např. chtěli použít bibliografii z jiného souboru, lze použít:

```
Make:biber {input = "jinysoubor"}
```

Příkazy `Make:image` a `Make:match` se spouští na výstupních souborech. Jejich prvním argument je regulární výraz, kterým se testují jména souborů, druhým je příkaz, který se spustí, pokud jméno souboru vyhovuje hledanému vzoru. Příkaz může být jednak šablona s programem, jednak Lua funkce.

Při konverzi obrázků se používají proměnné, které nejsou k dispozici v kompilačních příkazech, konkrétně `source` obsahuje jméno `.idv` souboru, `output` je název obrázku a `page` udává číslo stránky v `.idv` souboru.

Příkaz `Make:match` umožňuje měnit výstupní soubory. Proměnná `filename` obsahuje název výstupního souboru. V našem případě je tento příkaz spuštěn dvakrát, první příklad ukazuje použití filtrů, které jsou funkce v jazyce Lua. V tomto případě se jedná o filtr, který napravuje problém způsobený příkazem `\hrule`, jenž TEX4ht nemůže redefinovat, a v HTML souboru se projeví jako řada znaků `_`. Jako argument pro příkaz `filter` lze zadat více filtrů, kromě předdefinovaných, jako je `hruletohr`, lze zadat i vlastní funkce:

```
local changea = function(s) return s:gsub("a","z") end
local process = filter{"hruletohr", changea}
```

Tento příklad není úplně užitečný, neboť změní všechna písmena „a“ ve zpracovávaném dokumentu na „z“, včetně HTML tagů, ovšem jako ukázka snad postačí.

Pokud sestavovací soubor pojmenujete stejně jako název dokumentu s příponou `.mk4`, bude spouštěn automaticky, jinak ho lze zadat jako parametr `-e` pro TEX4ebook:

```
tex4ebook -f epub3 -e buildfile.mk4 filename.tex
```

Konfigurační soubor pro TEX4ht

Pro vložení vlastních HTML tagů do konfigurovatelných háčků nebo vlastních CSS instrukcí můžeme použít konfigurační soubor pro TEX4ht. Možnosti konfigurace si ukážeme na jednoduchém TEXovém souboru:

```
\documentclass{article}
\usepackage[T1]{fontenc}
```

```
\usepackage[utf8]{inputenc}
\usepackage[czech]{babel}
\begin{document}
Příliš žluťoučký kůň \textit{úpěl} \textbf{ďábelské ódy}
\end{document}
```

Po kompilaci se vygeneruje HTML soubor s následujícím obsahem:

```
<p class="noindent" >Příliš žluťoučký kůň <span
class="ecti-1000">úp</span><span
class="ecti-1000">ěl </span><span
class="ecbx-1000">ď</span><span
class="ecbx-1000">ábelsk</span><span
class="ecbx-1000">é </span><span
class="ecbx-1000">ódy </span>
```

Vygenerované soubory jsou ponechány i v adresáři s TEXovým dokumentem, lze je tedy snadno zkontrolovat ve WWW prohlížeči. Slova, která byla zvýrazněna jiným než základním řezem písma, byla rozdělena do několika HTML elementů. Ve vykresleném dokumentu se text zobrazí v pořádku, nicméně měli bychom tento problém vyřešit. Jedná se o chybu při konverzi `DVI` souboru příkazem `tex4ht`, který se snaží přidat formátování podle písma detekovaného v `DVI` souboru. Díky této funkcionalitě není třeba vytvářet konfigurace pro všechna uživatelská makra, základní textové formátování, jako jsou řezy a velikost písma, je zachováno. Pokud však TEX4ht narazí na znak s diakritikou, uzavře aktuální element, přestože nedochází ke změně písma. Tento problém lze vyřešit jednoduchou konfigurací pro dotčená makra v konfiguračním souboru pojmenovaném například `myconfig.cfg`:

```
\Preamble{xhtml}
\Configure{textbf}{\NoFonts\HCode{<strong>}}
{\HCode{</strong>}\EndNoFonts}
\Configure{textit}{\NoFonts\HCode{<em>}}
{\HCode{</em>}\EndNoFonts}
\begin{document}
\EndPreamble
```

Konfigurační soubor můžeme načíst volbou `-c`:

```
tex4ebook -f epub3 -e buildfile.mk4 -c myconfig.cfg filename.tex
```

Výsledný HTML kód již vypadá mnohem lépe:

```
<p class="noindent" >Příliš žluťoučký kůň
<em>úpěl</em> <strong>ďábelské ódy</strong>
```

Konfigurační soubory pro TEX4ht mají pevnou základní strukturu:

```
% zde můžeme vkládat balíčky
\Preamble{xhtml,volby pro tex4ht.sty}
..
```

```
% Konfigurace háčků a kaskádových stylů
\begin{document}
..
\EndPreamble
```

Před příkazem `\Preamble` můžeme nahrávat dodatečné balíčky pomocí příkazu `\RequirePackage`. Příkaz `\Preamble` umožňuje specifikovat volby, pro které ovlivňují nahrávání konfigurací. Jsou oddělené čárkami a povinná je minimálně volba `html` nebo `xhtml`, která musí být na první pozici. Pro Epub je třeba používat druhou z nich.

Po příkazu `\Preamble` můžete přidávat konfigurace pro háčky, kaskádové styly nebo měnit definice příkazů definovaných v preambuli dokumentu, ať již ve vkládaných balíčcích nebo přímo v dokumentu. Použití `\begin{document}` je povinné, definice, které po něm následují, jsou spuštěny až na začátku dokumentu. Konfigurace končí příkazem `\EndPreamble`.

V našem příkladu užíváme dvě konfigurace pro příkazy `\textbf` a `\textit`. Příkaz `\Configure` má variabilní počet argumentů, pro jednotlivé konfigurace je třeba konzultovat dokumentaci[2].

V tomto konkrétním případě užívá příkaz `\Configure` dva argumenty, jedná se o kód, který se vloží na začátek a na konec konfigurovaného textu. Příkazy `\NoFonts` a `\EndNoFonts` zakážou a povolí zpracování fontů z `DVI` souboru, čímž vyřešíme problém s rozdělenými slovy. Příkaz `\HCode` vkládá HTML kód do dokumentu.

Výsledný e-book vypadá poměrně stroze, pokud chcete vylepšit jeho vzhled, je možné použít kaskádové styly. Pro jednoduché úpravy je určen příkaz `\Css`, který lze použít v konfiguračním souboru:

```
\Preamble{xhtml}
% konfigurace písem vynechána
\Css{em{color:blue;}}
\begin{document}
```

Tímto příkazem obarvíme text v kurzívě modrou barvou.

Pro složitější změny je lepší vložit kompletní CSS styl. Existuje řada projektů stylů, které se zaměřují na dobrou typografii na webu. Můžeme využít například Scale.css (Salminen, 2012), který vytvořil finský designér Viljami Salminen. Tento styl je responzivní, což znamená, že velikost jednotlivých prvků dokumentu se přizpůsobuje velikosti zobrazované plochy tak, aby byla vždy optimální z hlediska dobré čitelnosti.

[2]Dokumentace není úplně silnou stránkou projektu TEX4ht. Základní informace o konfiguraci se nachází na WWW stránkách projektu, detailnější informace můžete získat z dokumentace vygenerované ze zdrojových kódů TEX4ht umístěných na stránkách `http://michal-h21.github.io/src4ht/` Jedná se především o dokumenty `tex4ht-info` a `tex4ht-html4`.

Pro vložení externího stylu můžeme využít příkaz `\AddCss`, který poskytuje balíček `include4ht`. Tento balíček je jeden z těch, které tvoří projekt `helpers4ht`. Jeho cílem je zjednodušit práci se systémem TEX4ht a obsahuje balíčky pro vkládání kaskádových stylů, JavaScriptových programů, tvorbu rejstříků a další. Balíček pro vkládání fontů můžeme použít pro vložení fontů do našeho e-booku.

Epub 3 podporuje vkládání písem ve formátu OpenType nebo jeho zjednodušené verzi určené pro web, WOFF. Je vhodné využívat OpenSource písma, vložením komerčního písma bychom se mohli dopustit jeho ilegálního šíření. Vhodná písma jsou například EB Garamond, Linux Libertine nebo Latin Modern. Pro každý řez písma je třeba stáhnout odpovídající OpenType soubor a umístit ho do adresáře s dokumentem, stejně tak je třeba umístit zde externí kaskádové styly. Upravený konfigurační soubor:

```
\RequirePackage{include4ht}
\RequirePackage{addfont4ht}
\Preamble{xhtml}
\AddCss{scale.css}
..
\NormalFont{EBGaramond}{EBGaramond12-Regular.woff}
\BoldFont{EBGaramond}{EBGaramond12-Italic.woff}
\ItalicFont{EBGaramond}{EBGaramond12-Italic.woff}
\Configure{@HEAD}{\HCode{<style type='text/css' >
    \Hnewline body{font-family:rmfamily,
      "EBGaramond", sans-serif;}\Hnewline
    </style>}}
\begin{document}
\EndPreamble
```

Příkazy `\NormalFont`, `\BoldFont` a `\ItalicFont` vloží fonty. První parametr určuje jméno rodiny, na které se dále odkazujeme. Příkaz `\Configure{@HEAD}` vkládá kód do hlavičky HTML dokumentu. V tomto případě je použit pro vložení kaskádového stylu, který vybírá námi definovanou rodinu *EBGaramond* jako hlavní písmo dokumentu.

Matematika a e-booky

Velký problém e-booků je matematika, stejně jako na webu, kde zobrazování matematiky řeší formát MathML. Alespoň teoreticky, protože jeho podpora ve WWW prohlížečích je špatná, všechny ho podporují pouze částečně.

Formát MathML je součástí specifikace Epub 3, teoreticky by ho tedy měla čtecí zařízení podporovat, realita je ovšem odlišná[3]. Pokud MathML podporují,

[3]Tabulku shrnující podporu v různých čtecích zařízeních lze nalézt na stránkách knihovny MathJax (The Mathjax Consortium, 2016).

je to za pomoci knihovny MathJax. Ta dokáže výborně vykreslit matematiku na WWW stránkách, ovšem je poměrně náročná na výkon a na tabletech nebo mobilních telefonech trvá delší chvíli, než se stránky obsahující matematiku vykreslí.

V případě ostatních formátů je situace ještě horší, v případě Epub i Mobi lze matematiku zobrazit pouze za pomoci obrázků, což pro TEX4ht není problém, naopak je to jeho výchozí nastavení. Problémem je, že tyto obrázky nebudou sedět na účaří a velikost a typ písma nebudou odpovídat okolnímu textu. Zatím jsem nenalezl uspokojivé a univerzální řešení těchto problémů, proto doporučuji pro matematické e-booky používat Epub 3 s podporou MathML, jehož podpora ve čtecích zařízeních se může zlepšit.

Podpora MathML se může zapnout pomocí přepínače `mathml` umístěného v příkazu `\Preamble` v konfiguračním souboru nebo ve druhém argumentu příkazu `tex4ebook`:

```
tex4ebook -f epub3 filename mathml
```

Konfigurace vlastních balíčků

Příkazem `\Configure` můžeme konfigurovat pouze příkazy, do kterých byly pro toto použití vložené háčky. Pokud chceme přidat podporu pro nový balíček, je třeba vytvořit pro něj `.4ht` soubor. Jeho tvorbu si ukážeme na jednoduchém příkladu.

Mějme balíček pro tvorbu vzdělávacích dokumentů, který umožňuje tvorbu dvou verzí dokumentu – pro studenty a pro učitele, s poznámkami, které se ve studentské verzi nezobrazují. Balíček bude mít přepínač, kterým se bude vybírat mezi zobrazeními, pro každou verzi bude vytvořen soubor, který bude sloužit pro kompilaci a který bude vkládat samotný text dokumentu.

Budeme také využívat rozšířených metadat, která poskytuje formát Epub 3. Vzniklo několik profilů, které specifikují metadata podle určení dokumentu. Pro tvorbu vzdělávacích dokumentů vznikl profil *Epub for Education* (The EPUB Worging Group, 2016), který využijeme v našem dokumentu.

Balíček `edupub.sty` je poměrně prostý:

```
\ProvidesPackage{edupub}
\RequirePackage{kvoptions}
\RequirePackage{etoolbox}
\newbool{teacher}
\boolfalse{teacher}
\newcommand\teacherinfo[1]{}
\DeclareVoidOption{teacher}{%
  \renewcommand\teacherinfo[1]{%
    \edupub@print@teacherinfo{##1}}
```

```
  \booltrue{teacher}}
\newcommand\edupub@print@teacherinfo[1]{#1}
\ProcessKeyvalOptions*
```

Definuje pouze příkaz `\teacherinfo`, který vypíše poznámky pro učitele, pokud je balíček nahrán s volbou `teacher`. Nyní vytvoříme konfigurační soubor pro TEX4ht, `edupub.4ht`:

```
\ifbool{teacher}{%
  \Configure{OpfMetadata}
  {\HCode{<dc:type>teacher-edition</dc:type>}}
}{}%
\NewConfigure{teacherinfo}{2}

\let\old:teacherinfo\edupub@print@teacherinfo
\renewcommand\edupub@print@teacherinfo[1]{
  \a:teacherinfo
  \old:teacherinfo{#1}
  \b:teacherinfo
}
\Configure{teacherinfo}
{\HCode{<span epub:type="answer">}}
{\HCode{</span>}}
```

Konfigurace `\Configure{OpfMetadata}` vloží informace do souboru s metadaty pro dokument. V tomto případě označí verzi dokumentu pro učitele.

Příkaz `\NewConfigure` deklaruje konfigurační háčky, které se vkládají do redefinovaných příkazů. První argument je název konfigurace, druhý je počet háčků, které budou vytvořeny. Pokud chceme vkládat nějaký kód před příkaz a po jeho skončení, stačí nám dva háčky. Háčky jsou příkazy ve formátu `\[a-i]:název háčku`, v našem případě jsou vytvořeny dva, `\a:teacherinfo` a `\b:teacherinfo`. Pomocí příkazu `\let` si můžeme uložit původní definici příkazu, který chceme upravit, a poté použít například příkaz `\renewcommand` pro úpravu definice.

Nakonec konfigurujeme naše háčky promocí `\Configure{teacherinfo}`, poznámky pro učitele označíme elementem `span` s atributem `epub:type`, který Epub 3 používá pro sémantické značkování textu. Tento konkrétní typ je součástí profilu *Epub for Education*.

Naše ukázková publikace tedy bude mít dvě verze, jednu pro studenty a druhou pro učitele. Pro každou vytvoříme řídicí dokument, který bude obsahovat pouze hlavičku dokumentu, samotný text se bude vkládat a bude pro obě verze stejný.

Řídicí soubor pro učitele `teacher.tex`:

```
\documentclass{article}
..
```

```
\usepackage[teacher]{edupub}
\begin{document}
\include{text}
\end{document}
```

Verze pro studenta bude totožná, pouze balíček `edupub` vložíme bez volby `teacher`. Samotný text dokumentu může mít následující obsah:

```
\newcommand\odpoved[1]{\teacherinfo{\\ #1}}
\begin{enumerate}
\item Jak se jmenuje největší had
      vyskytující se v~České republice?
\odpoved{Užovka stromová}
\end{enumerate}
```

Závěr

Článek představil základní použití nástroje pro tvorbu elektronických knih TEX-ebook, shrnul rozdíly mezi jednotlivými formáty elektronických knih a zaměřil se na jeden z nich, Epub 3. Tento formát je vhodný i pro publikaci technicky a matematicky zaměřených publikací, tudíž je zajímavý i pro uživatele TEXu. TEX4ebook byl úspěšně použit pro tvorbu materiálů pro výuku matematiky i několika menšími vydavateli.

Samotný TEX4ebook je nadstavba pro TEX4ht; pro upravení vzhledu výsledného dokumentu je možné využít konfigurační nástroje, které TEX4ht nabízí. Více informací lze nalézt v článku původního autora TEX4ht (Gurari, 2004) nebo ve čtvrté kapitole knihy *LATEX Web Companion* (Goosens a kol., 1999).

Reference

Drakos, Nikos. *LaTeX2HTML* [on-line]. *Ver. 98.1.* 2016. Dostupné na: `http://www.latex2html.org/`.

Goossens, Michel et al.. *The LATEX Web Companion: Integrating TeX, HTML, and XML.* Boston : Addison-Wesley Professional, 1999. 560 s. ISBN 978-0201433111.

Gurari, Eitan. *TeX4ht* [on-line]. *Ver. 200.* 1997. Dostupné na: `http://www.tug.org/tex4ht/`.

Gurari, Eitan. TEX4ht: HTML Production [on-line]. *TUGboat*, 2004, vol. 25, no. 1., s. 39–47. [cit. 2016-14-10]. (ISSN 0896-3207.) Dostupné na: `https://www.tug.org/TUGboat/tb25-1/gurari.pdf`.

Hoftich, Michal. *TEX4ebook* [on-line]. *Ver. 0.1d.* 2016a. Dostupné na: `https://www.ctan.org/pkg/tex4ebook`.

Hoftich, Michal. *Make4ht* [on-line]. *Ver. 0.1b.* 2016b. Dostupné na: https://www.ctan.org/pkg/make4ht.
Miller, Bruce. *LaTeXML* [on-line]. *Ver. 0.8.* 2016. Dostupné na: http://dlmf.nist.gov/LaTeXML/.
MacFarlane, John. *Pandoc* [on-line]. *Ver. 1.17.* 2016. Dostupné na: http://pandoc.org/.
Pettarin, Alberto. *Aeneas* [on-line]. *Ver. 1.6.* 2016. Dostupné na: https://www.readbeyond.it/aeneas/.
Pettarin, Alberto. *Menestrello* [on-line]. *Ver. 3.0.0.* 2016. Dostupné na: https://www.readbeyond.it/menestrello/index.html.
Salminen, Viljami. *Scale.css* [on-line]. *Ver. 0.1.* 2012. Dostupné na: https://github.com/viljamis/Scale.
The EPUB Working Group. *EPUB for Education* [on-line]. 2016. [cit. 2016-14-10]. Dostupné na: http://www.idpf.org/epub/profiles/edu/spec/.
The International Digital Publishing Forum. *EPUB 3* [on-line]. 2016. [cit. 2016-14-10]. Dostupné na: http://idpf.org/epub/30.
The MathJax Consortium. *EPUB3 Reading systems overview* [on-line]. 2016. [cit. 2016-14-10]. Dostupné na: http://docs.mathjax.org/en/latest/misc/epub.html.

Summary: E-books and the TEX4ebook System

This article describes the process of conversion of a TEX document to e-book using TEX4ebook system. Concrete examples of configuration and caveats are provided.

Keywords: e-books, TeX4ht, HTML, conversion.

Michal Hoftich, michal.h21@gmail.com

Sadzba bibliografie podľa normy ISO 690 v systéme LaTeX

Dávid Lupták

Dodržiavanie normy ISO 690 pri tvorbe bibliografických odkazov a citácií býva vyžadované mnohými inštitúciami nielen v českom akademickom prostredí. V systéme LaTeX však doteraz neexistovala žiadna podpora, ktorá by plnohodnotne riešila túto problematiku. Až na základe referenčnej implementácie balíka biblatex-iso690 vznikol balíček, ktorý spĺňuje požiadavky normy v plnom rozsahu a výrazne tak zjednodušuje citovanie informačných zdrojov.

Kľúčové slová: ISO 690, bibliografia, citácie, BibLaTeX

1. Úvod

Odborné články vyžadujú vysoký stupeň práce s inými odbornými textami a informačnými zdrojmi, na ktoré je potrebné korektne sa odkazovať a tieto zdroje správne citovať. V českom akademickom prostredí prevláda tvorba bibliografických odkazov a citácií podľa normy ISO 690 [1]. Úvodom tohoto článku je v stručnosti predstavená norma ISO 690 a takisto aj pre ňu už existujúce implementácie. Osobitne sú uvedené možnosti sadzby bibliografie v systéme LaTeX a v záverečnej časti nasleduje oboznámenie sa s balíkom biblatex-iso690, prvou úplnou LaTeXovou podporou aktuálne platnej normy ISO 690.

2. Norma ISO 690

Tvorba bibliografických odkazov a citácií sa v minulosti riadila pravidlami noriem ISO 690:1987 [2] pre tlačené informačné zdroje a ISO 690-2:1997 [3] pre zdroje elektronické. V roku 2010 boli tieto dve normy zjednotené a nahradené novou verziou normy ISO 690:2010 [4]. Normalizačné organizácie (členovia ISO) zabezpečujú preklady noriem na národnej úrovni [5]. Takýmto prekladom bola v roku 2011 prijatá aj česká verzia normy ČSN ISO 690:2011 [6] alebo v roku 2012 slovenská technická norma STN ISO 690:2012 [7]. Tieto preklady nadobúdajú rovnaký status ako oficiálna verzia normy ISO 690:2010. Aj napriek tomu obsahuje napríklad česká norma ČSN ISO 690:2011 množstvo nejasných miest a nepresností. Táto problematika je však prenechaná na iné články Zpravodaja [8].

 doi: 10.5300/2016-1-4/106

2.1. Terminológia

K úspešnému porozumeniu nasledovného textu je potrebné uviesť dva základné termíny, ktoré častokrát bývajú medzi sebou zamieňané[1] [6]:

odkaz údaj v texte alebo iný druh obsahu dokumentu odkazujúci na príslušnú bibliografickú citáciu

citácia dáta popisujúce informačný zdroj alebo jeho časť dostatočne presne a podrobne na to, aby mohol byť tento zdroj identifikovaný a bolo možné ho vyhľadať

2.2. Zásada konzistencie

Norma ISO 690 hneď v úvode svojho výkladu priamo uvádza, že nemá za cieľ definovať konkrétny štýl bibliografického odkazu alebo citácie. Použitý štýl a interpunkcia v ilustračných príkladoch nie sú súčasťou doporučenia. Tento fakt prináša dva zásadné poznatky:

1. norma ctí princíp oddelenia formy od obsahu
2. normu nemožno považovať za citačný štýl [8]

Zároveň však norma doporučuje, aby bol pre všetky citácie v dokumente použitý jednotný štýl, formát a interpunkcia. Táto požiadavka však už zostáva na samotnom upravovateľovi citácie, ktorý môže čerpať z ilustračných príkladov v norme samotnej, interpretácií alebo iných zaužívaných zvyklostí sadzby bibliografie.

3. Sadzba bibliografie v systéme LaTeX

V systéme LaTeX sú k dispozícii prakticky tri základné prístupy na sadzbu bibliografie [11]. Prvým z nich je použitie čistého LaTeXu, zvyšné dva uznávajú princíp oddelenia formy od obsahu a na sadzbu bibliografie používajú externú bibliografickú databázu a takisto aj externý program na preklad.

3.1. Čistý LaTeX

LaTeX poskytuje na sadzbu bibliografie vstavané prostredie `thebibliography` a rodinu makier `\cite` umožňujúcich sadzbu bibliografických odkazov. Tie následne odkazujú na samotné bibliografické citácie. Zoznam bibliografických citácií je uvedený v prostredí `thebibliography`, jednotlivé položky sú uvedené príkazom `\bibitem`.

```
\documentclass{...}
\begin{document}
```

[1] Potvrdzuje to aj článok o predstavení originálu normy, ktorý bol napísaný ešte pred vznikom českej verzie normy [9]; takisto interpretácia normy od tej istej autorky [10].

```
\cite{<identifikátor01>}
...
\begin{thebibliography}{<najširší identifikátor>}
\bibitem{<identifikátor01>}
  <Autor>. \emph{<Názov>: <podnázov>}. ...
...
\end{thebibliography}
\end{document}
```

Predchádzajúca ukážka kódu okrem základnej syntaxe poukazuje aj na fakt, že takýto prístup nie je vhodný na sadzbu bibliografie rozsiahlejších diel [12]. Prináša tieto hlavné nevýhody:

1. vysádzané sú všetky citácie uvedené vnútri prostredia `thebibliography` (bez ohľadu na to, či boli vôbec citované)
2. každý záznam je nutné naformátovať jednotlivo (v závislosti od požadovaného bibliografického štýlu)
3. bibliografické citácie sú radené presne v takom poradí ako sú uvedené v zozname prostredia `thebibliography`

Okolnosti ohľadom nevýhody číslo 1 norma ISO 690 priamo nešpecifikuje, takýto prístup však nenasleduje všeobecné odporúčania tvorby bibliografie [11]. Ďalej je v prípade prostredia `thebibliography` ťažké zaručiť zásadu jednotnosti citácií (podľa 2) a vôbec nie je možné zaistiť správne poradie citácií (podľa 3) pre ktorúkoľvek prípustnú metódu citovania špecifikovanú v norme ISO 690.

Znovupoužiteľnosť bibliografických záznamov a škálovateľnosť zoznamu citácií nie je silnou stránkou tohoto riešenia. Výhodou je však rýchlosť a nízky počet prekladov dokumentu (postačí dvojnásobný preklad TEXovým kompilátorom).

3.2. BibTEX

Preferovaným spôsobom práce s bibliografiou pri sadzbe rozsiahlejších typov dokumentov je vytvorenie externej databázy bibliografických záznamov (pozri podsekciu 3.4) a použitie externého programu na jej preklad [11]. Tento externý program zaistí správne zoradenie citácií (rieši problém 3) a podľa použitého bibliografického štýlu[2] (rieši problém 2) vygeneruje LATEXové prostredie `thebibliography` s bibliografiou na vysadenie. Typickým zástupcom tohoto prístupu je BIBTEX, ktorého nespornou výhodou je práve spomínané oddelenie formy od obsahu.

Príkaz `\bibliographystyle` slúži na definovanie formátovacieho štýlu, príkaz `\bibliography` určuje, ktoré bibliografické databázy sa majú použiť a takisto miesto vysadenia v zdrojovom dokumente. Príkazom `\cite` sa vytvorí odkaz

[2] BIBTEXový bibliografický styl je často nazývaný aj citačný štýl; v tomto kontexte by bolo možné hovoriť všeobecne o formátovacom štýle.

v texte dokumentu na danú citáciu. Je možné použiť aj príkaz `\nocite`, ktorý nevytvorí odkaz v texte samotného dokumentu, zaručí však výskyt citácie v zozname bibliografických citácií (rieši problém 1).

```
\documentclass{...}
\bibliographystyle{<formátovací štýl>}
\begin{document}
\cite[<text>]{<zoznam identifikátorov>}
...
\bibliography{<databáza01>,<databáza02>,...}
\end{document}
```

Spomenuté výhody však vyvažuje, resp. prevyšuje, množstvo nevýhod spojených so skutočnosťou, že vývoj BibTeXu má relatívne stagnujúcu tendenciu [13, 14]. Medzi hlavné nevýhody patrí:

1. problémy so vstupným kódovaním [15] (hoci je dostupné alternatívne riešenie)[3]
2. zložitosť tvorby vlastných formátovacích štýlov [16] (hoci je dostupné riešenie na automatizované generovanie štýlov)[4]
3. výkonnostné problémy (pretečenie pamäti pri práci s veľkými bibliografickými databázami) [17]
4. slabá podpora tvorby odkazov v texte dokumentu [18] (hoci sú dostupné flexibilnejšie možnosti)[5]
5. absencia dnes už štandardných polí, napr. poľa `url` (hoci sú dostupné alternatívne riešenia)[6]
6. chýbajúca podpora lokalizácie a viacjazyčnosti citácií [19] (hoci je riešenie dostupné)[7]

Na korektné vysádzanie zdrojového dokumentu sú potrebné minimálne tri preklady TeXovým kompilátorom a jeden programom BibTeX. Globálne aplikovateľná schéma na sadzbu bibliografie pomocou BibTeXu je nasledovná [20]:

$$\text{LaTeX}\ (\text{BibTeX}\ \text{LaTeX})^{+}\ \text{LaTeX}$$

Oproti sadzbe bibliografie bez použitia externého programu je síce počet potrebných prekladov vyšší, na druhej strane však táto „zložitosť“ prináša riešenia na takmer všetky vyššie spomenuté problémy spojené s použitím iba čistého LaTeXu.

[3] `https://www.ctan.org/pkg/bibtex8bit`
[4] `https://www.ctan.org/pkg/custom-bib`
[5] `https://www.ctan.org/pkg/natbib`, `https://www.ctan.org/pkg/cite`
[6] `https://www.ctan.org/pkg/natbib`, `https://www.ctan.org/pkg/babelbib`
[7] `https://www.ctan.org/pkg/babelbib`

3.3. BibLaTeX

Inou možnosťou externého programu na sadzbu bibliografie je modernejší a flexibilnejší LaTeXový balík BibLaTeX. Tento balíček je kompletnou reimplementáciou prostriedkov na prácu s bibliografiou v systéme LaTeX, často označovaný aj ako nástupca BibTeXu [15, 21]. Na formátovanie záznamov používa výhradne LaTeXové makrá a na spracovanie bibliografickej databázy (pozri podsekciu 3.4) a jednotlivých položiek nástroj Biber [17].

Použitie balíka BibLaTeX je mierne odlišné od tradičného BibTeXu. Príkazy majú nielen odlišnú syntax, ale poskytujú aj rozsiahlejšie možnosti práce s bibliografickými záznamami. Formátovací štýl (je možné špecifikovať bibliografický a citačný štýl zvlášť) sa definuje priamo ako voľba balíka pri jeho načítaní, t. j. ako voliteľný argument príkazu `\usepackage`. Na špecifikovanie bibliografickej databázy slúži príkaz `\addbibresource`, kde je potrebné uviesť celý názov súboru aj s príponou `.bib`. Na vysádzanie samotnej bibliografie v texte slúži príkaz `\printbibliography`, ktorý sa v dokumente môže vyskytovať *aj* viackrát. Samozrejme na tvorbu odkazu v texte dokumentu slúži príkaz `\cite` a jeho varianty. Nasledujúca ukážka kódu pokrýva len základnú štruktúru dokumentu, použitie príkazov je omnoho komplexnejšie:

```
\documentclass{...}
\usepackage[...]{biblatex}
\addbibresource{<databáza01.bib>}
\addbibresource{<databáza02.bib>}
\begin{document}
\cite{...}
...
\printbibliography
\end{document}
```

Balík BibLaTeX rieši množstvo problémov uvedených v prípade BibTeXu. Medzi tie najvýznamnejšie patrí [22]:

1. plná podpora kódovania Unicode
2. pokročilé možnosti radenia (použité technológie *Unicode Collation Algorithm* a *Unicode Common Locale Data Repository* (CLDR))
3. viacjazyčnosť záznamov (balíky babel a polyglossia)
4. rozšírený formát bibliografických záznamov
5. veľké množstvo dostupných štýlov
6. flexibilné vytváranie nových štýlov

Samozrejme, toto je len výňatok z bohatej funkcionality BibLaTeXu [17]. Dôkazom toho je aj minimálny počet nevýhod tohoto balíka. Za všetky je možné spomenúť nekompatibilitu medziformátu sadzby bibliografie s pôvodným BibTeXovým riešením [23], ktorý býva vyžadovaný pri predkladaní vydavateľovi.

Preklad prebieha analogicky ako u BibTeXu. Najskôr sa prekladá TeXovým kompilátorom, následne sa použije spomínaný nástroj Biber nad vygenerovaným súborom `.bcf` a nakoniec je potrebný ešte jeden preklad kompilátorom TeXu. Schéma sadzby dokumentu za použitia BibLaTeXu vyzerá nasledovne[8]:

```
latex <dokument>[.tex]
biber <dokument>[.bcf]
latex <dokument>[.tex]
```

3.4. Databáza bibliografických záznamov (`.bib` súbor)

Pre úplnosť tejto sekcie je žiadúce uviesť aj samotný formát bibliografickej databázy. Tá obsahuje jednotlivé bibliografické záznamy, každý takýto záznam má svoj typ, jedinečný identifikátor a (typicky) niekoľko dvojíc *pole–hodnota* definujúcich práve samotné bibliografické údaje. Všeobecný vzor takéhoto záznamu vyzerá nasledovne:

```
@<typ záznamu>{identifikátor,
  <názov poľa> = {hodnota},
  ...
  <názov poľa> = {hodnota}
}
```

Všetky podporované typy záznamov BibTeXu sú aplikovateľné aj pre balík BibLaTeX, a to buď priamo alebo definovaním aliasu. Rozdielom je akurát to, že BibLaTeX oproti BibTeXu definuje naviac ešte niekoľko ďalších typov.

Obdobná situácia je aj v prípade polí záznamov. Balík BibLaTeX poskytuje spätnú kompatibilitu pre všetky polia dostupné v BibTeXu a k tomu ponúka ďalšie. Okrem *bežných* polí sú to aj polia *špeciálne*, ktoré slúžia napríklad na nastavenie jazyka daného bibliografického záznamu pre podporu viacjazyčnosti.

3.5. Zhrnutie

Na jednoduchú sadzbu bibliografie samostatného dokumentu s malým počtom citácií je najvýhodnejšie použiť vstavanú funkcionalitu LaTeXu. Pri rozsiahlejších dokumentoch s väčším počtom citácií je však už výhodné použiť služby externých programov na sadzbu bibliografie. Takýmto prístupom oddelenia formy od obsahu (bibliografická databáza sa nachádza v samostatnom súbore, formátovacie štýly taktiež osobitne) je možné dosiahnuť vysokú škálovateľnosť, znovupoužiteľnosť bibliografických záznamov a flexibilnú a efektívnu manipuláciu s citáciami.

[8] Koncovky súborov nie je potrebné uvádzať.

Okrem samotného BibTeXu existuje množstvo programov na ňom založených, ich problémom však je práve skutočnosť, že vychádzajú z BibTeXu. Týka sa to najmä použitia formátovacích štýlov, hoci niektoré sa pokúšajú o nahradenie jazyka BST[9] iným, modernejším, programovacím jazykom (zväčša XML) [21, 24].

Spomedzi dostupných možností sadzby bibliografie v systéme LaTeX [11, 25] vychádza ako najlepšia voľba balík BibLaTeX s nástrojom Biber [21, 26].

4. Existujúce riešenia pre ISO 690

Táto sekcia uvádza niekoľko existujúcich implementácií, ktoré zahŕňajú podporu pre dodržiavanie normy ISO 690. Prvé dve z nich sú priamo použiteľné v sádzacom systéme LaTeX, jazyk CSL je predstavený z dôvodu jeho súčasnej popularity a balík OPmac-bib ako jeden z mála dostupných balíkov poskytujúcich plnohodnotnú podporu tvorby bibliografických odkazov a citácií.

4.1. czechiso

Pre české normy ČSN ISO 690:1996 [27] a ČSN ISO 690-2:2000 [28] existuje pre BibTeX neoficiálny formátovací štýl `czechiso` z roku 2006. Autorom je David Martinek a tento štýl je dostupný na adrese `http://www.fit.vutbr.cz/~martinek/latex/czechiso.html`. Táto implementácia nezodpovedá normám presne, niektoré vyžadované položky bibliografických záznamov absentujú, použité funkcie by bolo vhodné prepísať aby splňovali požiadavky normy.

4.2. biblatex-iso690

V roku 2011 vznikla prvá referenčná implementácia bibliografického a citačného štýlu pre balík BibLaTeX podľa normy ISO 690. Tá však vychádzala z predošlých verzií noriem [27, 28] a prevažne sa pridržiavala českej interpretácie [29]. Autorom je Michal Hoftich a štýl bol sprístupnený ako neoficiálna verzia na adrese `https://github.com/michal-h21/biblatex-iso690`. Podobne ako v predchádzajúcom prípade, ani táto implementácia nezodpovedala normám presne, na domovskej stránke projektu je zaznamenaných niekoľko nahlásených problémov ohľadom funkcionality a použitia tohoto štýlu. V súčasnosti však vďaka kompletnej revízii poskytuje plnohodnotnú podporu normy ISO 690 (pozri podsekciu 5).

4.3. Jazyk CSL

Jazyk CSL (*Citation Style Language*) je programovací jazyk založený na jazyku XML. Populárnym sa stal s vydaním Zotera v roku 2006 [30].

[9] bibliografický štýl BibTeXu

Medzi jednoznačné výhody patrí syntax jazyka XML. Na to nadväzuje obľúbenosť tohoto formátu a otvorenosť a univerzálnosť jazyka CSL [31]. Nespornou výhodou je aj jeho využiteľnosť naprieč viacerými aplikáciami, čo dokazuje aj rozsiahly zoznam produktov na oficiálnych stránkach projektu CSL, ktoré tento jazyk využívajú [32]. Patrí medzi ne napríklad Zotero, Papers či Mendeley.

V repozitári projektu *Citation Style Language* [32] sú medzi obrovským množstvom citačných štýlov dostupné aj tie podľa normy ISO 690, v celkovom počte 15 štýlov. Z tohoto počtu je každý pre inú jazykovú lokalizáciu alebo metódu citovania. V každom z nich sa nájdu aj drobné odchýlky alebo nepresnosti voči výkladu normy ISO 690. Vo všeobecnosti existuje pre jazyk CSL tento rad nevýhod [33]:

- nie je možné nastaviť formát bibliografických odkazov
- obmedzená podpora pre legislatívne štýly (Multilingual Zotero môže byť riešením)
- obmedzená podpora pre viacjazyčnosť citácií (Multilingual Zotero môže byť riešením)
- nie je možné zadať rozsah dátumu do poľa pre dátum (údaj sa nevygeneruje)

Na záver je potrebné dodať, že napríklad balík BibLaTeX všetky tieto problémy pokrýva vo svojej základnej funkcionalite [17].

4.4. OPmac-bib

OPmac je sada makier umožňujúca pohodlnejšiu prácu s plainTeXom, ktoré poskytujú základnú LaTeXovú funkcionalitu. Balíček OPmac-bib je nadstavbou týchto makier, zaoberajúci sa práve sadzbou bibliografie. Oproti ostatným nástrojom však nepoužíva žiaden externý program, ale celú funkcionalitu rieši na úrovni TeXových makier. Autorom balíka je Petr Olšák a k dispozícii je od roku 2015 v rámci balíka csplain. Viac detailov je prenechaných na článok TUGboatu [34].

OPmac-bib pracuje priamo s `.bib` súbormi a je akousi nadstavbou a rozšírením nad starodávnym BibTeXom. Využíva teda všetky typy a polia, ktoré sú dostupné v BibTeXu a naviac prináša nové polia, ktoré sú v dnešnej dobe nutnosťou (vo všeobecnosti ako aj pre štýl ISO 690). Ide napríklad o polia `url`, `doi` alebo `lang`, vďaka ktorým už nemusí autor `.bib` súboru vkladať tieto informácie do poľa `note`, ale do príslušných, na tento účel vytvorených, polí. Tým sa zvyšuje flexibilita a v konečnom dôsledku aj možnosť korektného výpisu poradia údajov podľa aktuálnej verzie normy.

Tým, že balíček OPmac-bib úplne obchádza použitie externého programu a číta `.bib` databázu priamo pomocou makier TeXu, zvyšuje čitateľnosť kódu a zároveň umožňuje jednoduchšie predefinovanie makier v prípade nutnosti ich prispôsobenia špecifickým potrebám. Norma ISO 690 totiž vynucuje uvádzanie niektorých údajov, pre ktoré v pôvodnom BibTeXu príslušné polia neexistujú. Vytváranie nových polí pre každý takýto špecifický údaj však nie je riešenie. OPmac to z tohoto hľadiska rieši vskutku elegantne, obdobne ako balík BibLaTeX.

Poskytuje univerzálne polia, do ktorých možno uvádzať nielen bibliografické údaje, ale zároveň aj makrá používané na výpis prvkov a tým docieliť požadovanú podobu výstupu. Ide napríklad o polia `option` a `ednote`.

Pole `option` je možné použiť napríklad v prípade potreby uvedenia dodatočného názvu diela, jeho prekladu apod. Vďaka takýmto voľbám je možné dosiahnuť adekvátny výpis údajov spĺňajúci aktuálnu verziu normy ISO 690.

Pole `ednote` slúži na uvedenie vedľajších tvorcov alebo na rôzne doplňujúce informácie. Bežne sa tu môžu vyskytnúť údaje, ktoré nemožno jednoducho algoritmizovať, preto je potrebné do tohoto poľa zadať údaje v takej podobe, v akej sa majú objaviť na výstupe. Typickým príkladom môžu byť informácie o prekladateľovi alebo pôvodcoch ďalších vydaní.

Vďaka flexibilným poliam záznamov poskytuje balík OPmac-bib plnú podporu normy ISO 690, a preto je jasnou voľbou pre použitie pri práci s plainTEXom.

5. Balík biblatex-iso690

Spomedzi existujúcich riešení uvedených v predošlej sekcii je pre sadzbu v LaTeXu relevantný jedine balík biblatex-iso690. Existujúce odchýlky od normy ISO 690 boli z pôvodnej implementácie odstránené a v súčasnosti balík vyhovuje pravidlám poslednej verzie normy ISO 690 pre tvorbu bibliografických odkazov a citácií. Viac nástrojov a služieb je popísaných v bakalárskej práci autora tohoto článku [35].

Pôvodný stav balíka biblatex-iso690 obsahoval nasledovné chyby a nedostatky:

- pridržiavanie sa predošlých verzií noriem
- chybné poradie prvkov citácie
- nadbytočná/chýbajúca interpunkcia
- chýbajúca podpora niektorých typov dokumentov
- chýbajúca podpora niektorých vyžadovaných prvkov citácie
- chýbajúca sekundárna zodpovednosť
- zastaraný kód

Pri revízii štýlu biblatex-iso690 napokon došlo k úplnej reimplementácii celého štýlu. Zmenou a korekciou prešli nielen časti kódu, ktoré mali za účel vypisovať jednotlivé elementy citovanej jednotky v správnom poradí. Boli to aj všetky pomocné makrá, príkazy a definície zabezpečujúce korektné spracovávanie a formátovanie bibliografických údajov (z bibliografickej databázy – `.bib` súboru). Viaceré požiadavky normy bolo možné zabezpečiť jednoducho priamo pomocou poskytovanej množiny funkcií a vďaka možnostiam balíka BibLaTeX. Niektoré však bolo nutné vyriešiť pozmenením základných makier a príkazov BibLaTeXu. Požiadavky, ktoré nebolo možné zabezpečiť algoritmicky, príp. zostali ponechané na tvorcu bibliografickej `.bib` databázy:

- chýbajúca podpora citovania formou priebežných poznámok
- zalamovanie URL adries

- algoritmické riešenie (ne)uvádzania prvého vydania publikácie
- alg. riešenie (ne)uvádzania iba jedného (primárne prvého) vydavateľa
- alg. riešenie (ne)uvádzania iba jedného (primárne prvého) miesta vydania
- termín *Anon* pre anonymné diela
- lokalizačný reťazec `nodate` pre neznámy rok vydania

5.1. Metódy citovania

Norma ISO 690 predpisuje tri metódy citovania informačných zdrojov. Okrem už spomenutej metódy formou priebežných poznámok je to tzv. harvardská metóda (označovaná aj ako metóda autor-dátum) a tzv. numerická metóda (forma číselného odkazu). V balíku biblatex-iso690 sú tieto metódy dostupné ako samostatné štýly pod názvom `iso-authoryear`, resp. `iso-numeric`. Tieto štýly sa špecifikujú už v preambule dokumentu pri zavádzaní balíka BibLaTeX, napr. `\usepackage[style=iso-numeric]{biblatex}`.

5.2. Voľby balíka – prispôsobiteľnosť

Norma ISO 690 nepredpisuje pre bibliografické citácie konkrétny štýl, formát a interpunkciu. Na zmenu výstupného formátu citácie sú k dispozícii voľby balíka biblatex-iso690 a príp. je možné aj predefinovanie v preambule dokumentu. Aktuálne dostupné voľby sú:

- `spacecolon=[true|false]` na zmenu výpisu dvojbodky pri názve a podnázve diela alebo pri nakladateľských informáciách
  - Miesto : Nakladateľstvo
  - Miesto: Nakladateľstvo
- `pagetotal=[true|false]` pre zobrazenie celkového počtu strán ako doplňujúcej informácie
  - Miesto: Nakladateľstvo, 2008 [60 s.]
  - Miesto: Nakladateľstvo, 2008
- `shortnumeration=[true|false]` pre typograficky odlíšený skrátený výpis informácií o číslovaní
  - ... 2011, **32**(3), 289–301 [cit. 2016-05-14] ...
  - ... 2011, roč. 32, č. 3, s. 289–301 [cit. 2016-05-14] ...
- `thesisinfoinnotes=[true|false]` určenie poradia informácií o záverečnej kvalifikačnej práci
  - ... Dostupné z: <...>. BP. MU, FI, Brno. Vedúci práce Petr SOJKA
  - ... BP. MU, FI, Brno. Vedúci práce Petr SOJKA. Dostupné z: <...>

5.3. Integrácia do šablóny fithesis3

Fithesis3 je oficiálna šablóna Masarykovej univerzity na sadzbu záverečných kvalifikačných prác v LaTeXu [36]. Trieda je navrhnutá s možnosťou jednoduchej

rozšíriteľnosti i pre iné akademické inštitúcie. Preto bolo prirodzenou požiadavkou začleniť aj balík biblatex-iso690 do štýlu fithesis3 s cieľom maximalizovať používateľskú prívetivosť. Integrácia do šablóny fithesis3 prebiehala v spolupráci s hlavným vývojárom tohoto balíka – Vítom Novotným – a spočíva v nasledovných krokoch:

- nový kľúč `bib` vo voľbách balíka fithesis3, do ktorého sa špecifikujú bibliografické databázy (`.bib` súbory)
- následné automatické zavedenie metódy citovania podľa zvolenej fakulty
- automatická sadzba zoznamu bibliografických citácií na konci dokumentu
- samozrejme je možná ručná špecifikácia vyššie uvedeného (pozri podsekciu 3.3)

```
\documentclass{fithesis3}
\thesissetup{
  ...
  bib = {<bibliografická-databáza.bib>}
  ...
}
\begin{document}
\cite{...}
...
\end{document}
```

5.4. Dostupnosť

Ako už bolo spomenuté, pre systém LaTeX doposiaľ neexistovala žiadna oficiálna podpora tvorby bibliografických odkazov a citácií podľa normy ISO 690. Až revíziou balíka biblatex-iso690 vznikla jeho oficiálna podoba, ktorá je teraz dostupná na medzinárodnom archíve CTAN ako balík biblatex-iso690. Zároveň je pod rovnakým menom dostupný aj v TeXovej distribúcii TeX Live 2016.

6. Záver

Článok sa zaoberá sadzbou bibliografie v systéme LaTeX podľa normy ISO 690. Úvodom je načrtnutá problematika samotnej normy ISO 690 a následne sú predstavené i možnosti sadzby bibliografie v systéme LaTeX. Na základe existujúcich riešení s podporou normy ISO 690 je bližšie predstavený balík biblatex-iso690, ktorý sa po iniciálnej implementácii v roku 2011 dočkal úplnej reimplementácie dodržujúc pravidlá poslednej revízie normy ISO 690. Bibliografické citácie uvedené v tomto článku sú vysádzané za použitia implementovaného balíka biblatex-iso690, môžu teda slúžiť ako referenčný zoznam bibliografických citácií.

Zoznam bibliografických citácií

1. KRATOCHVÍL, Jiří; SEJK, Petr; ELIÁŠOVÁ, Věra; STEHLÍK, Marek. *Metodika tvorby bibliografických citací* [online]. 2. revidované vydání. Návrh obálky MAZOCH, Břetislav. Brno: Masarykova univerzita, 2011 [cit. 2016-03-16]. ISSN 1802-128X. Dostupné z: `http://is.muni.cz/do/rect/el/estud/prif/ps11/metodika/web/ebook_citace_2011.html`.
2. *ISO 690: Information and documentation – Bibliographic references – Content, form and structure*. Second edition. Geneva: The International Organization for Standardization, 1987.
3. *ISO 690-2: Information and documentation – Bibliographic references – Part 2: Electronic documents or parts thereof*. First edition. Geneva: The International Organization for Standardization, 1997.
4. *ISO 690: Information and documentation – Guidelines for bibliographic references and citations to information resources*. Third edition. Geneva: The International Organization for Standardization, 2010.
5. INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO Membership Manual* [online]. Geneva, 2015 [cit. 2016-05-20]. ISBN 978-92-67--10611-3. Dostupné z: `http://www.iso.org/iso/iso_membership_manual.pdf`.
6. *ČSN ISO 690: Informace a dokumentace – Pravidla pro bibliografické odkazy a citace informačních zdrojů*. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 2011. Třídící znak 01 0197.
7. *STN ISO 690: Informácie a dokumentácia. Návod na tvorbu bibliografických odkazov na informačné pramene a ich citovanie*. Bratislava: Úrad pre normalizáciu, metrológiu a skúšobníctvo Slovenskej republiky, 2012. Triediaci znak 01 0197.
8. HÁLA, Tomáš. Komentář k nové revizi normy ČSN ISO 690 – Pravidla pro bibliografické odkazy a citace informačních zdrojů. *Zpravodaj Československého sdružení uživatelů TEXu*. 2013, roč. 23, č. 2, s. 107–112. ISSN 1211-6661. Dostupné z DOI: `10.5300/2013-2/107`.
9. BRATKOVÁ, Eva. Co přináší třetí vydání mezinárodní normy ISO 690:2010. *Ikaros* [online]. 2010, roč. 14, č. 9 [cit. 2016-05-19]. ISSN 1212-5075. Dostupné z: `http://ikaros.cz/node/13524`.
10. BRATKOVÁ, Eva. *Citování informačních zdrojů a tvorba bibliografických referencí podle mezinárodní normy ISO 690:2010* [online]. Praha: Ústav informačních studií a knihovnictví FF UK v Praze, 2011. Verze 1.0 [cit. 2016-03-16]. Dostupné z: `http://www1.cuni.cz/~brt/bibref/bibref.html`.
11. TALBOT, Nicola L. C. *Using LaTeX to Write a PhD Thesis*. Norfolk, UK: Dickimaw Books, 2013. Dickimaw LaTeX Series. ISBN 978-1-909440-02-9.
12. TALBOT, Nicola L. C. *LaTeX for Complete Novices*. Norfolk, UK: Dickimaw Books, 2012. Dickimaw LaTeX Series. ISBN 978-1-909440-00-5.

13. PATASHNIK, Oren. BibTeX 1.0. *TUGboat: Proceedings of the 1994 Annual Meeting* [online]. 1994, roč. 15, č. 3, s. 269–273 [cit. 2016-05-14]. Dostupné z: https://www.tug.org/TUGboat/tb15-3/tb44patashnik.pdf.
14. PATASHNIK, Oren. BibTeX Yesterday, Today, and Tomorrow. *TUGboat: Proceedings of the 2003 Annual Meeting* [online]. 2003, roč. 24, č. 1, s. 25–30 [cit. 2016-05-14]. Dostupné z: https://www.tug.org/TUGboat/tb24-1/patashnik.pdf.
15. BibTeX: Process bibliographies for LaTeX, etc. *CTAN: The Comprehensive TeX Archive Network* [online]. 2010 [cit. 2016-05-14]. Dostupné z: https://www.ctan.org/pkg/bibtex.
16. PATASHNIK, Oren. *Designing BibTeX Styles* [online]. 1988 [cit. 2016-05-14]. Dostupné z: http://mirrors.ctan.org/biblio/bibtex/base/btxhak.pdf.
17. LEHMAN, Philipp; WRIGHT, Joseph; BORUVKA, Audrey; KIME, Philip. *The BibLaTeX package: Programmable Bibliographies and Citations* [online]. 2016. Verzia 3.4 [cit. 2016-03-16]. Dostupné z: http://mirrors.ctan.org/macros/latex/contrib/biblatex/doc/biblatex.pdf.
18. SHELL, Michael; HOADLEY, David. *BibTeX Tips and FAQ* [online]. 2007. Verzia 1.1 [cit. 2016-05-14]. Dostupné z: http://mirrors.ctan.org/biblio/bibtex/contrib/doc/btxFAQ.pdf.
19. HARDERS, Harald. Multilingual Bibliographies: Using and extending the babelbib package. *TUGboat* [online]. 2002, roč. 23, č. 3/4, s. 344–353 [cit. 2016-05-14]. Dostupné z: https://www.tug.org/TUGboat/tb23-3-4/tb75harders.pdf.
20. MARKEY, Nicolas. *Tame the BeaST: The B to X of BibTeX* [online]. 2009. Verzia 1.4 [cit. 2016-05-14]. Dostupné z: http://tug.ctan.org/tex-archive/info/bibtex/tamethebeast/ttb_en.pdf.
21. HUFFLEN, Jean-Michel. A comparative study of methods for bibliographies. *TUGboat: TUG 2011 Proceedings* [online]. 2011, roč. 32, č. 3, s. 289–301 [cit. 2016-05-14]. Dostupné z: https://www.tug.org/TUGboat/tb32-3/tb102hufflen.pdf.
22. BibLaTeX: Sophisticated Bibliographies in LaTeX. *CTAN: The Comprehensive TeX Archive Network* [online]. 2016 [cit. 2016-05-14]. Dostupné z: https://www.ctan.org/pkg/biblatex.
23. BibLaTeX: submitting to a journal. *TeX – LaTeX Stack Exchange* [online]. 2011 [cit. 2016-05-14]. Dostupné z: http://tex.stackexchange.com/questions/12175/biblatex-submitting-to-a-journal.
24. HUFFLEN, Jean-Michel. Languages for bibliography styles. *TUGboat: Proceedings of the 2008 Annual Meeting* [online]. 2008, roč. 29, č. 3, s. 401–412 [cit. 2016-05-14]. Dostupné z: https://www.tug.org/TUGboat/tb29-3/tb93hufflen.pdf.

25. MITTELBACH, Frank; GOOSSENS, Michel; BRAAMS, Johannes; CARLISLE, David; ROWLEY, Chris. *The LaTeX Companion*. Second Edition. Boston: Addison-Wesley, 2004. Tools and Techniques for Computer Typesetting. ISBN 0-201-36299-6. Fourth printing (with corrections), Sept. 2005.
26. KIME, Philip; CHARETT, François. *Biber: A backend bibliography processor for biblatex* [online]. 2016. Verzia 2.5 [cit. 2016-05-14]. Dostupné z: `http://mirrors.ctan.org/biblio/biber/documentation/biber.pdf`.
27. *ČSN ISO 690: Dokumentace – Bibliografické citace – Obsah, forma a struktura*. Praha: Český normalizační institut, 1996. Třídící znak 01 0197.
28. *ČSN ISO 690-2: Informace a dokumentace – Bibliografické citace – Část 2: Elektronické dokumenty nebo jejich části*. Praha: Český normalizační institut, 2000. Třídící znak 01 0197.
29. BRATKOVÁ, Eva (zost.). *Metody citování literatury a strukturování bibliografických záznamů podle mezinárodních norem ISO 690 a ISO 690-2: metodický materiál pro autory vysokoškolských kvalifikačních prací* [online]. Verze 2.0, aktualiz. a rozšíř. Praha: Odborná komise pro otázky elektronického zpřístupňování vysokoškolských kvalifikačních prací, Asociace knihoven vysokých škol ČR, 2008 [cit. 2016-02-02]. Dostupné z: `http://www.evskp.cz/SD/4c.pdf`.
30. FENNER, Martin. Citation Style Language: An Interview with Rintze Zelle and Ian Mulvany. *Gobbledygook* [online]. 2010 [cit. 2016-04-16]. Dostupné z: `http://blogs.plos.org/mfenner/2010/09/24/citation-style-language-an-interview-with-rintze-zelle-and-ian-mulvany/`.
31. ANSORGE, Libor; KRATOCHVÍL, Jiří. Popis šablony ČSN ISO 690:2011 v jazyce CSL pro citační manažer Zotero. *ProInflow*. 2013, roč. 5, č. 2. ISSN 1804-2406. Dostupné také z: `http://pro.inflow.cz/sites/default/files/pdfclanky/Kratochvil_Ansorge_Sablona_0.pdf`.
32. ZELLE, Rintze. *CitationStyles.org | The Citation Style Language: open and free citation styles* [online]. 2016 [cit. 2016-05-17]. Dostupné z: `http://citationstyles.org/`.
33. ZELLE, Rintze. Styles. *CitationStyles.org* [online]. 2016 [cit. 2016-05-17]. Dostupné z: `http://citationstyles.org/styles/`.
34. OLŠÁK, Petr. OPmac-bib: Citations using `*.bib` files with no external program. *TUGboat* [online]. 2016, roč. 37, č. 1, s. 71–78 [cit. 2016-12-31]. Dostupné z: `https://www.tug.org/members/TUGboat/tb37-1/tb115olsak-bib.pdf`.
35. LUPTÁK, Dávid. *Sazba bibliografie dle normy ISO 690* [online] [cit. 2016-06-14]. Dostupné z: `http://is.muni.cz/th/422640/fi_b/`. BP. Masarykova univerzita, Fakulta informatiky, Brno. Vedoucí práce Petr SOJKA.
36. NOVOTNÝ, Vít; MAREK, Daniel; PAVLOVIČ, Jan; SOJKA, Petr. *The fithesis3 class for the typesetting of theses written at the Masaryk University in Brno* [GIT]. 2015 [cit. 2016-05-16]. Dostupné z: `http://github.com/witiko/fithesis3.git`.

Summary: Typesetting Bibliographies Compliant with the International Standard ISO 690 in LaTeX

The preparation of bibliographic references and citations compliant with the international standard ISO 690 is required by many institutes not limited to the Czech and Slovak academia. However, the typesetting of bibliographies conforming to the respective standard is not yet supported in the LaTeX document preparation system. The biblatex-iso690 package has been revised and improved to fully meet the requirements of the international standard and thus greatly simplifies the typesetting of bibliographies for all kinds of information resources.

Keywords: ISO 690, bibliography, reference, citation, BibLaTeX

Dávid Lupták, 422640@mail.muni.cz
Fakulta informatiky, Masarykova univerzita, Brno

Mělo by to fungovat V – Cykly

Peter Wilson

Abstrakt

Článek ukazuje, jak je možné v LaTeXu procházet řetězce a zpracovávat je znak po znaku. Dále článek popisuje LaTeXové makro `\@for` a jeho použití ukazuje na příkladu sazby tabulky.

Klíčová slova: LaTeX, řetězce, cykly, `\@for`.

> 'Tis better to be lowly born
> And range with humble livers in content
> Than to be perked up in a glist'ring grief
> And wear a golden sorrow.
>
> *Henry VIII*
> William Shakespeare

Cílem tohoto seriálu je ukázat čtenáři krátké kousky kódu, které mohou vyřešit některé z jeho problémů. Doufám, že situaci ještě více nezkomplikuji v důsledku mých chyb. Opravy, poznámky a návrhy na změny budou vždy vítány.

> To no one but the Son of Heaven does it belong to order ceremonies, to fix the measures, and to determine the written characters.
>
> *The Analects*
> Confucius

1. Práce s řetězci

Ve svém dřívějším článku (Wilson, 2005) jsem se zmínil, že se ještě později vrátím k práci s řetězci. Následující makra se mohou použít k postupnému zpracování všech znaků v jednoduchém řetězci.

```
\chardef\catcodeG=\catcode`\^^G
\catcode`\^^G=12
\newcommand*{\vsechnyznaky}[1]{%
  \def\argument{#1}\ifx\argument\@empty\else
  \vsechny@znaky#1^^G\fi}
```

Z anglického originálu *Glisterings* (Wilson, 2007) přeložil Jan Šustek.

doi: 10.5300/2016-1-4/121

```
\def\vsechny@znaky#1#2^^G{%
  \def\znak{#1}\def\dalsi{#2}%
  \ifx\znak\@empty\let\next\@gobble
  \else
    \zpracujznak{#1}%
    \ifx\dalsi\@empty \let\next\@gobble
    \else \let\next\vsechny@znaky \fi
  \fi
  \next#2^^G}
\catcode`\^^G=\catcodeG
```

V makrech jsem použil speciální znak `^^G` jako oddělovač konce řetězce. Za normálních okolností TEX bere tento znak jako neplatný (a některé balíčky jej mění na aktivní znak), ale zde jsem dočasně změnil jeho kategorii na „ostatní“ znak. Makro `\@gobble` je v LATEXu definováno jako makro, které vezme jeden argument a nic s ním neudělá. Makro `\vsechnyznaky` při své expanzi zavolá makro `\zpracujznak{`⟨*znak*⟩`}` pro každý znak v řetězci. Makro `\zpracujznak` můžeme definovat například následovně.

```
\newcommand*{\zpracujznak}[1]{\textit{#1}}
```

Potom makro `\vsechnyznaky` dá následující výsledky.

```
\vsechnyznaky{vsechnyznaky}                    vsechnyznaky
\vsechnyznaky{{\oe}rsted}                      œrsted
\vsechnyznaky{}
\vsechnyznaky{slova s mezerami}                slovasmezerami
```

Speciální případ prázdného argumentu řeší přímo makro `\vsechnyznaky`, zatímco ostatní případy jsou předány makru `\vsechny@znaky`. Toto makro volá samo sebe a přitom postupně načítá jednotlivé znaky původního řetězce. Tomuto procesu se říká *řetězová rekurze*, což znamená, že poslední věc, kterou makro udělá, je, že zavolá samo sebe, případně neudělá nic, pokud se má rekurze ukončit.

Připomínám, že pokud v LATEXu píšete zdrojový kód, ve kterém se vyskytují makra obsahující zavináč, pak je musíte psát uvnitř balíčku (v souboru .sty), nebo je musíte vložit mezi makra `\makeatletter` a `\makeatother`.

Nepříjemnou vlastností makra `\vsechnyznaky` je, že pohltí všechny mezery v zadaném řetězci. V následujícím řešení mezery zpracujeme ve dvou krocích. Nejdříve projdeme řetězec po jednotlivých slovech, kde za slovo považujeme posloupnost znaků oddělenou mezerou. Ve druhém kroku projdeme slovo po jednotlivých znacích.

Nejdříve si nastavíme potřebné věci a definujeme hlavní makro `\vsechnaslova`.

```
\newif\if@zacatekslova
\def\textrelax{\relax}
```

```
\chardef\catcodeG=\catcode`\^^G
\catcode`\^^G=12
\newcommand*{\vsechnaslova}[1]{%
  \vsechna@slova#1^^G}
\def\vsechna@slova#1^^G{%
  \nactislovo#1 ^^G}
```

Makro `\nactislovo` oddělí následující slovo od zbytku řetězce (argument je oddělený mezerou). Pak zavolá makro `\nactiznak`, jehož argumentem je ono slovo.

```
\long\def\nactislovo#1 {%
  \@zacatekslovatrue
  \nactiznak#1\relax}
```

Makro `\nactiznak` načte následující „znak“ slova. Pokud je tento „znak“ `^^G`, znamená to konec celého řetězce. Pokud je tento znak „`\relax`“, znamená to konec slova a musí se znovu zavolat makro `\nactislovo` na zbytek řetězce. Ve zbývajících případech se na tento znak aplikuje makro `\zpracujznak` a znovu se zavolá makro `\nactiznak`.

```
\def\nactiznak#1{%
  \def\znak{#1}%
  \if\znak^^G\let\next\relax
  \else
    \ifx\znak\textrelax
      \let\next\nactislovo
    \else
      \zpracujznak{#1}%
      \let\next\nactiznak
    \fi
  \fi
  \next}
\catcode`\^^G=\catcodeG
```

Makro `\nactiznak` je dalším příkladem použití řetězové rekurze.

Nyní makro `\zpracujznak` nadefinujeme trochu komplikovaněji. Makro otestuje, zda se znak nachází na začátku slova. Pokud ano, pak se znak převede na velké písmeno a vysází se kurzívou. Pokud ne, pak se znak vysází tučně. Tento příklad asi v praxi moc nevyužijeme, nicméně je to ukázka toho, jakým způsobem můžeme zpracovávat znaky v řetězci.

```
\newcommand*{\zpracujznak}[1]{%
  \if@zacatekslova
    \space\textit{\MakeUppercase{#1}}%
```

```
    \@zacatekslovafalse
  \else \textbf{#1}\fi}
```

Následuje několik příkladů.

```
\vsechnaslova{retezec s mezerami}
\vsechnaslova{{\oe}rsteduv zakon}
```

Retezec *S* *M*ezerami
Œrsteduv *Z*akon

Tato makra budou fungovat dobře pro jednoduché řetězce, ale velmi pravděpodobně budou dávat nesprávné výsledky, pokud řetězce budou obsahovat akcentované znaky nebo jiný materiál narušující plynulé čtení znaků. Na druhou stranu, asi by bylo jednodušší a rychlejší změnit řetězce ručně v textovém editoru.

Here we go loop de loop.
Here we go loop de li.
Here we go loop de loop
On a Saturday night.

Loop de loop
Johnny Thunder?

2. Cykly

Občas potřebujeme opakovaně provádět akci, která se netýká zpracování řetězců. V TEXu pro tyto účely existují makra `\loop`...`\repeat`. Používají se následovně

```
\loop
  ⟨příkazy⟩
\if ⟨podmínka⟩
  ⟨další příkazy⟩
\repeat
```

TEX nejdříve vykoná ⟨*příkazy*⟩ a pak vyhodnotí ⟨*podmínku*⟩ (u tohoto příkazu `\if` se nesmí vyskytovat `\else` ani `\fi`). Pokud je ⟨*podmínka*⟩ splněna, TEX provede ⟨*další příkazy*⟩, vrátí se zpět a pokračuje znovu ⟨*příkazy*⟩. Pokud ⟨*podmínka*⟩ není splněna, přeskočí TEX ⟨*další příkazy*⟩ a pokračuje za `\repeat`.

LATEX poskytuje mechanismus, který umožňuje procházet seznam položek oddělených čárkami (jako například seznam voleb třídy nebo balíčku). Obecná syntaxe je

```
\@for\promenna:=⟨seznam⟩\do{⟨příkazy⟩}
```

kde ⟨*seznam*⟩ obsahuje hodnoty oddělené čárkami. V cyklu se do makra `\promenna` postupně uloží jednotlivé hodnoty a s každou se provedou ⟨*příkazy*⟩.

Jak cyklus pracuje, si ukážeme na příkladu. Následuje očesaná verze maker z třídy memoir (Wilson, 2004). Makra umožňují zarovnat seznam věcí do tabulky, aniž bychom se zabývali ukončováním řádků. Hlavní makro

58 `\fillrows{`⟨*šířka*⟩`}{`⟨*počet*⟩`}{`⟨*seznam*⟩`}`

vytvoří vycentrovanou tabulku o celkové šířce ⟨*šířka*⟩, která má ⟨*počet*⟩ sloupců a v jednotlivých buňkách jsou položky ⟨*seznamu*⟩. Položky se do tabulky zapíšou v přirozeném pořadí, tj. zleva doprava a potom shora dolů. Nápad na makro `\fillrows` jsem dostal po přečtení *TEX for the Impatient* (Abrahams a kol., 1990), kde byla popsána verze pro TEX, která tabulku vyplňovala v pořadí shora dolů a potom zleva doprava.

Většina následujícího kódu je podobná kódu ukrytému v definici prostředí `tabular` a nebudu se snažit tento kód vysvětlovat.

Nejdříve nadeklarujeme čítače a délkové registry, které budeme potřebovat.

```
\newcount\CT@cols      % počet sloupců
\newcount\@cellstogo   % počet zbývajících sloupců
\newdimen\CT@col@width % šířka sloupce
\newtoks\crtok
  \crtok = {\cr}%
```

Nyní již můžeme definovat makro `\fillrows`. Makro má tři argumenty – celkovou šířku, počet sloupců a seznam položek. Na začátku makra nastavíme čítače podle počtu sloupců.

```
\newcommand{\fillrows}[3]{\par\begingroup
  \CT@cols=#2\relax
  \@cellstogo=\CT@cols
```

Další část kódu se provede po vložení každé položky do tabulky. Buď vloží `&`, nebo alternativu makra `\\`.

```
  \def\@endcolactions{%
    \global\advance\@cellstogo\m@ne
    \ifnum\@cellstogo<\@ne
      \global\@cellstogo=\CT@cols
      \the\crtok
    \else
      &
    \fi}%
```

Dále se spočítá šířka sloupců a nastaví se rozvržení tabulky.

```
  \CT@col@width=#1
  \divide\CT@col@width \CT@cols
  \penalty 10000\relax
  \noindent
  \vskip -\z@
  \def\@preamble{}%
```

```
81  \begingroup
82    \let\@sharp\relax
```

Nyní pomocí cyklu `\loop`...`\repeat` projdeme všechny sloupce kromě posledního a vždy přidáme k makru `\@preamble` vhodné mezery a znak `&`.

```
83    \ifnum\CT@cols>\@ne
84      \loop
85        \g@addto@macro{\@preamble}{%
86          \hb@xt@ \CT@col@width
87            {\strut\relax\@sharp\hfil} &}%
88        \advance\CT@cols\m@ne
89      \ifnum\CT@cols>\@ne
90      \repeat
91    \fi
```

Za posledním sloupcem znak `&` nebude.

```
92    \g@addto@macro{\@preamble}{%
93      \hb@xt@ \CT@col@width
94        {\strut\relax\@sharp\hfil}}%
95  \endgroup
```

(Výše uvedený kód nastaví šířku každého sloupce na hodnotu `\CT@col@width`. Pokud zakomentujeme řádky 86 a 93, bude šířka každého sloupce nastavena na přirozenou hodnotu podle jeho nejširší položky.[1]) Nyní již můžeme dokončit přípravné práce na tabulce.

```
96  \let\@sharp ##
97  \tabskip\fill
98  \halign to\hsize \bgroup
99    \tabskip\z@
100   \@preamble
101   \tabskip\fill\cr
```

Jednotlivé položky se do tabulky vloží cyklem `\@for`.

```
102   \@for\@tempa:=#3\do{%
103     \@tempa\unskip\space\@endcolactions}%
```

Na tomto místě již je tabulka hotová a můžeme ukončit definici makra `\fillrows`.

```
104   \the\crtok \egroup \endgroup \par}
```

Jako jednoduchý příklad vytvoříme následující tabulky.

[1] Potom samozřejmě bude první argument makra `\fillrows` nepodstatný, jako je tomu v příkladu na řádcích 107–109. (pozn. překl.)

```
\fillrows{0.5\textwidth}{4}{jedna,dvě,tři,čtyři,pět,%
  šest,sedm,osm,devět,deset}
```

jedna	dvě	tři	čtyři
pět	šest	sedm	osm
devět	deset		

```
\fillrows{0.5\textwidth}{7}{A,tady,je,výsledek,použití,makra,%
  \tt\char`\\fillrows,při,sazbě,do,sedmi,sloupců,nastavených,%
  na,svou,přirozenou,šířku.}
```

A	tady	je	výsledek	použití	makra	`\fillrows`
při	sazbě	do	sedmi	sloupců	nastavených	na
svou	přirozenou	šířku.				

Reference

Abrahams, Paul W., Berry, Karl, Hargreaves, Kathryn A. *TEX for the Impatient.* Boston, MA, USA : Addison-Wesley, 1990. 357 s. ISBN 0-201-51375-7. Dostupné na CTAN v adresáři `info/impatient/`.

Wilson, Peter. *The memoir class for configurable typesetting.* 2004. Dostupné na CTAN v adresáři `latex/macros/contrib/memoir/`.

Wilson, Peter. Glisterins [on-line]. *TUGboat*, 2005, **26**(3), s. 253–255. [cit. 2017-01-09]. (ISSN 0896-3207.) Dostupné na: `https://www.tug.org/TUGboat/tb26-3/tb84glister.pdf`.

Wilson, Peter. Glisterins [on-line]. *TUGboat*, 2007, **28**(1), s. 12–14. [cit. 2017-01-09]. (ISSN 0896-3207.) Dostupné na: `https://www.tug.org/TUGboat/tb28-1/tb88glister.pdf`.

Summary

This paper shows how to use process strings, character by character, in LaTeX. The paper also describes LaTeX macro `\@for` and shows its application for typesetting tables.

Keywords: LaTeX, strings, loops, `\@for`.

Peter Wilson, herries.press@earthlink.net
18912 8th Ave. SW
Normandy Park, WA 98166 USA

Informace o Zpravodaji CSTUG

Zpravodaj CSTUG je recenzovaný vědecký a odborný časopis s původními pracemi vycházející od r. 1991. Od roku 1997 je tištěná verze opatřena číslem ISSN 1211-6661. Od roku 2002 vychází časopis i elektronicky, on-line verzi bylo přiděleno ISSN 1213-8185. Periodikum je vydáváno v Brně.

Profil periodika: Zpravodaj CSTUG se zaměřuje na problematiku typografického systému TEX, jeho implementaci a vývoj, na jeho následníky (LATEX, pdfTEX, XETEX, LuaTEX aj.) a na příbuzné a podpůrné programové vybavení. Dále se též zabývá problematikou typografie obecně, zpracováním textu, sazbou a souvisejícími tématy, s cílem podporovat využití systému, zejména mezi českými a slovenskými uživateli. Přijímají se též texty z příbuzných oblastí, jako jsou například knižní produkce, nakladatelská a vydavatelská činnost, polygrafická produkce, technologie apod.

Zpravodaj CSTUG slouží ke sdílení nových poznatků, postupů a metod. Uveřejňuje původní práce, projektová řešení a práce přehledové i osvětové.

Časopis nabízí publikační příležitost jak renomovaným odborníkům, tak i začínajícím autorům. Uveřejněny mohou být rovněž i zkrácené verze závěrečných prací stejně jako výstupy projektů grantových agentur.

Detailní pokyny pro autory: Podrobné informace se nacházejí na `http://www.cstug.cz/pokyny/`.

Šablonu v TEXu pro autory pro snazší přípravu zdrojového textu článku naleznete na `http://bulletin.cstug.cz/`.

Složení redakční rady je uváděno v tiráži každého čísla a dále na webu Zpravodaje CSTUG, viz `https://www.cstug.cz/redakcni-rada/`.

Kontaktovat redakční radu můžete prostřednictvím e-mailu `zpravodaj@cstug.cz`, pro kontakt s **administrací sdružení** slouží `secretary@cstug.cz`.

Zpravodaje CSTUG vydává od jeho založení Československé sdružení uživatelů TEXu, z. s., IČ 005 36 580. Evidenční číslo registrace vedené Ministerstvem kultury dle zákona č. 46/2000 Sb. je E 7629. Bližší informace o vydavateli naleznete na webu sdružení `https://www.cstug.cz/kontakty`.

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Antonín Strejc
Ilustrace na obálce:	Bogusław Jackowski, Piotr Strzelczyk a Piotr Pianowski
Počet výtisků:	310
Uzávěrka:	31. 12. 2016
Odpovědný redaktor:	Jan Šustek
Redakční rada:	Pavel Haluza, Lukáš Novotný, Vít Novotný, Michal Růžička a Jan Šustek (šéfredaktor)
Technická redakce:	Vít Novotný
Technická spolupráce:	Tomáš Hála
Evidenční číslo MK:	E 7629
Tisk:	ASMETI, Klášterní 1187, 735 11 Orlová
Adresa:	CSTUG, c/o FEL ČVUT, Technická 2, 166 27 Praha 6
Email:	cstug@cstug.cz

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`, `zpravodaj@cstug.cz`
korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
korespondence předsedovi sdružení

`gacstug@cstug.cz`
grantová agentura CSTUGu

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD a DVD

`cstug-members@cstug.cz`
korespondence členům sdružení

`cstug-faq@cstug.cz`
řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`bookorders@cstug.cz`
objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
`ftp://ftp.cstug.cz`

www server sdružení:
`http://www.cstug.cz`

CONTENTS