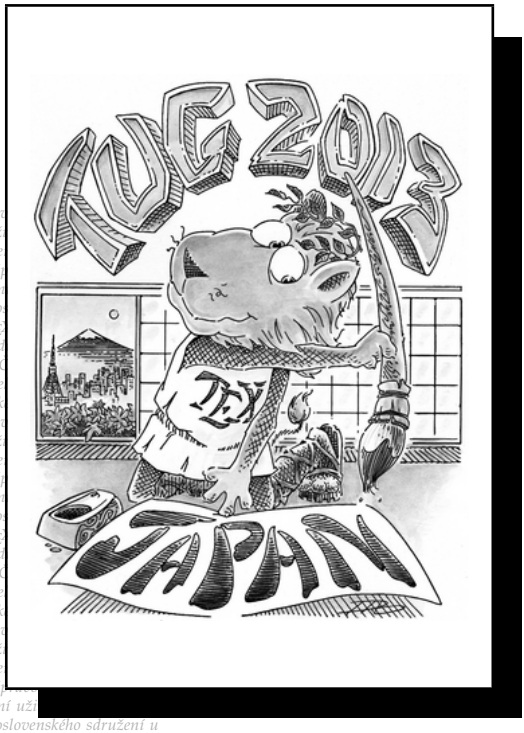
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

3

2012

ISSN 1211-6661 Tištěná verze

ISSN 1213-8185 Online verze

Ročník 22

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/`.

Zpravodaj je zařazen do Seznamu recenzovaných neimpaktovaných periodik vydávaných v České republice, viz `http://www.vyzkum.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/`. Pokud nemáte přístup na Internet, můžete zaslat příspěvek na disketě, CD, či DVD na adresu:

Zdeněk Wagner
Vinohradská 114
130 00 Praha 3

`zpravodaj@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)

Pozvánka na TUG 2013

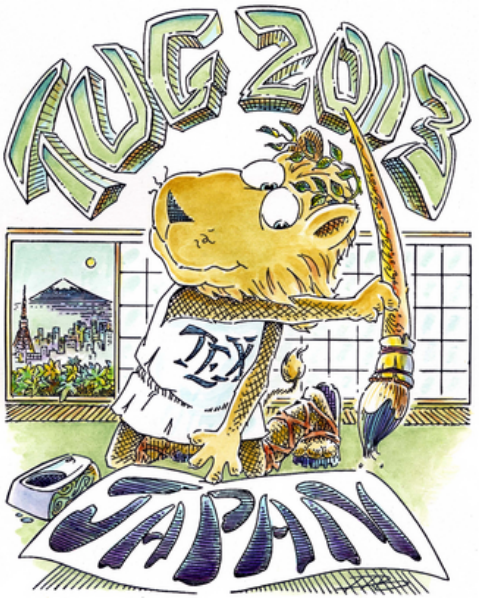
Mezinárodní sdružení uživatelů TEXu, `http://www.tug.org/`, Vás srdečně zve na 34. konferenci TUG 2013, která se letos bude konat v Japonsku na půdě Tokijské univerzity (東京大学). Uskuteční se 23. – 26. října 2013.

Webové stránky:
`http://tug.org/tug2013/`, příp. `http://tug.org/tug2013/jp/`

Konferenční email:
`tug2013@tug.org`

Důležité termíny:

- 15. července: žádost o finanční příspěvek od TUGu, a taktéž,
- 15. července zaslání návrhu prezentací a včasná registrace,
- 9. září: příprava tiskové verze programu,
- 22. října: slavnostní zahájení konference,
- 23. – 26. října: vlastní konference,
- 26. října: slavnostní večer,
- 4. listopadu: zaslání podkladů pro sborníkové číslo časopisu TUGboat.



Mohlo by to fungovat. II – Formuláře

Peter Wilson

Abstrakt:

Příspěvek ukazuje kousky LaTeXového kódu, které mohou autorovi usnadnit sazbu formulářů. Jsou zde uvedena makra na tvorbu různých zaškrtávacích políček a linek a ukázáno využití LaTeXového prostředí picture k sazbě celého formuláře.

Klíčová slova: LaTeX, sazba formulářů, prostředí picture

> Not all that tempts your wand'ring eyes
> And heedless hearts, is lawful prize;
> Nor all, that glisters, gold.
>
> *Ode to a Favourite Cat*
> Thomas Gray

Cílem tohoto seriálu je ukázat čtenáři krátké kousky kódu, které mohou vyřešit některé z jeho problémů. Sám jsem se naučil, že nejrychlejším způsobem, jak na diskusní skupině `comp.text.tex` dostat správnou odpověď, je napsat nesprávnou odpověď. Doufám, že níže uvedené myšlenky budou fungovat vždy, ale jak už název naznačuje, mezi zrníčky zlata mohou být i zrníčka písku.

Opravy, poznámky a návrhy na změny budou vždy vítány.

Jedna z méně častých otázek na `comp.text.tex` je, zda existuje nějaký balíček na sazbu formulářů, a odpověď je, že neexistuje. Přesto se formuláře v LaTeXu sázejí.

> Little boxes on the hillside, little boxes made of ticky tacky,
> Little boxes, little boxes, little boxes all the same.
>
> *Little boxes*
> Malvina Reynolds, 1961
> (Popularizoval Pete Seeger)

Přeložil Jan Šustek.

doi: 10.5300/2012-3/130

1. Zaškrtávací políčka

Jednou z běžných součástí formulářů jsou zaškrtávací políčka.
Umíte vytvořit zaškrtávací políčko? Ano ☐ Ne ☐
Umím vytvořit zaškrtávací políčko? Ano ☒ Ne ☐

Prázdná políčka byla vytvořena níže uvedeným makrem `\tickbox` a zaškrtnuté políčko makrem `\xbox`.

```
\newcommand*{\tickbox}{{\fboxsep 0pt%
  \framebox[\height]{\vphantom{M}}}}
\newcommand*{\xbox}{{\fboxsep 0pt%
  \framebox[\height]{\vphantom{M}$\times$}}}
```

Pokud chceme větší políčko ☐, použijeme místo makra `\tickbox` makro `\Tickbox`.

```
\newcommand*{\Tickbox}{\framebox{\phantom{M}}}
```

Registr `\fboxsep` určuje vzdálenost mezi textem uvnitř `\framebox` a rámečkem kolem něj, takže rozměry políčka můžeme nastavit změnou `\fboxsep` nebo použitím jiného znaku uvnitř `\phantom`.

Pokud preferujeme čtvercová políčka ☐, můžeme použít makro `\TickBox`.

```
\newcommand*{\TickBox}{{\fboxsep 0pt%
  \fbox{\rule{0em}{1em}\rule{1em}{0em}}}}
```

Tato definice používá neviditelné linky (`\rule` s nulovou výškou nebo šířkou není vidět) k nastavení šířky `\fbox`u. V tomto případě je políčko čtvercové, protože linky mají stejnou délku.

2. Prázdná místa, pomlčky a linky

Kromě zaškrtávacích políček je další běžnou součástí formulářů _____.

Na konci poslední věty jsme použili `\hrulefill.`, čímž jsme dostali linku roztaženou až na konec řádku. Makro `\hrulefill` můžeme použít více než jednou na řádku, podobně jako na následujícím řádku, kde bylo použito dvakrát.
Jméno: _____ Příjmení: _____

Můžeme chtít, aby linka měla danou délku. Linka na dalším řádku je dlouhá přesně tak, aby se na ni vešel „nějaký text“.
Napište _____ zde.
Napište nějaký text zde.

Předchozí dva řádky byly vysázeny pomocí

```
Napište \underline{\phantom{nějaký text}} zde.\\
Napište nějaký text zde.
```

Následuje několik dalších linek a prázdných míst. Vždy je ukázán zdrojový text a výsledek.

1. `Vyplňte \rule{10mm}{0.4pt} prázdné místo.`
   Vyplňte _____ prázdné místo.
2. `Vyplňte \hrulefill{} prázdné místo.`
   Vyplňte _____ prázdné místo.
3. `Vyplňte \xfill[0.5ex] prázdné místo.`
   Vyplňte _____ prázdné místo.
4. `Vyplňte \srule{něčím} prázdné místo.`
   Vyplňte _____ prázdné místo.
5. `Vyplňte \srule[0.5ex]{něčím} prázdné místo.`
   Vyplňte _____ prázdné místo.
6. `Vyplňte \phantom{něčím} prázdné místo.`
   Vyplňte prázdné místo.

Výše uvedená makra jsou součástí LaTeXu, s výjimkou maker `\xfill` a `\srule`, která jsou definována níže.

Makro `\xfill[`⟨*délka*⟩`]` je podobné makru `\hrulefill`. Makro nakreslí do volného prostoru linku a posune ji o hodnotu volitelného argumentu ⟨*délka*⟩ nahoru nebo dolů.

```
\newcommand*{\xfill}[1][0pt]{%
  \cleaders
    \hbox to 1pt{\hss
      \raisebox{#1}{\rule{1.2pt}{0.4pt}}%
      \hss}\hfill}
```

Makro `\srule[`⟨*délka*⟩`]{`⟨*text*⟩`}` nakreslí linku stejně širokou jako ⟨*text*⟩, ale ⟨*text*⟩ nevysází. Volitelný argument ⟨*délka*⟩ se používá k posunutí linky nahoru nebo dolů.

```
\newcommand*{\srule}[2][0pt]{%
  \setbox0\hbox{#2}%
  \rule[#1]{\wd0}{0.4pt}}
```

Příkaz `\rule` má volitelný argument, který určuje, o kolik bude linka posunuta nad nebo pod účaří textu. LaTeX poskytuje dvě pomlčky, krátkou (–), která se zapisuje `--`, a dlouhou (—), která se zapisuje `---`. Krátké pomlčky se používají u číselných rozsahů, například 2–4. V závislosti na národních typografických tradicích se krátké nebo dlouhé pomlčky používají jako interpunkce místo čárky. Delší pomlčky mohou označovat, že něco chybí; dvoučtverčíková pomlčka (——)

Customs Declaration **CD 44**
May be opened officially

☒ Gift ☐ Commercial sample
☐ Documents ☐ Other

Quantity and detailed description of contents	Weight lb.	oz.	Value
Toy			15
Scarf			12
For commercial items only *If known,* HS tariff number and country of origin of goods	Total Weight		Total Value

I, the undersigned, whose name and address are given on the item, certify that the particulars given in this declaration are correct and that this item does not contain any dangerous article or articles prohibited by legislation or by postal or customs regulations.

Date and sender's signature

PS Form **1234**, March 2004

pro chybějící písmena ve slově a tříčtverčíková (———) pro chybějící slovo. Pro tyto delší pomlčky můžete použít příkaz `\rule`.

```
\newcommand*{\iiemdash}{% dvoučtverčíková pomlčka
  \rule[0.5ex]{2em}{0.4pt}}
\newcommand*{\iiiemdash}{% tříčtverčíková pomlčka
  \rule[0.5ex]{3em}{0.4pt}}
```

3. Formuláře

Zjistil jsem, že často nejjednodušším způsobem, jak vysázet formulář, je použití prostředí `picture`, které umožňuje umístit věci přesně tam, kde chci. Zde je možná nudný příklad celního prohlášení.

```
\newcommand{\form}{%
\setlength{\unitlength}{1mm}
\begin{picture}(79,80)
\sffamily \scriptsize \thicklines
\put(0,0){\line(1,0){80}}
\put(0,5){\line(1,0){80}}
```

```
\put(2,4){\makebox(0,0)[tl]{\normalsize PS Form
          \textbf{1234}, March 2004}}
\put(0,14){\line(1,0){80}}
\put(2,13){\makebox(0,0)[tl]{Date and sender's
           signature}}
\put(0,26){\line(1,0){80}}
\put(2,25){\makebox(0,0)[tl]{%
  \begin{minipage}{76mm}
  I, the undersigned, ... regulations.
  \end{minipage}}}
\put(0,30){\line(1,0){48}}
\put(0,39){\line(1,0){80}}
\put(2,38){\makebox(0,0)[tl]{%
  \begin{minipage}{44mm}
  \textbf{For commercial items only} \\
  \textsl{If known,} ... \end{minipage}}}
\put(56,38){\makebox(0,0)[t]{Total Weight}}
\put(72,38){\makebox(0,0)[t]{Total Value}}
\put(0,53){\line(1,0){80}}
\put(2,52){\makebox(0,0)[tl]{%
  \begin{minipage}{40mm}
  \CONT \end{minipage}}}
\put(0,60){\line(1,0){80}}
\put(2,59){\makebox(0,0)[tl]{%
  \begin{minipage}{40mm}
  Quantity ... \end{minipage}}}
\put(49,59){\makebox(0,0)[tl]{%
  \begin{minipage}{14mm}
  \hfill Weight \hfill \mbox{}\\
  lb. \hfill oz. \end{minipage}}}
\put(72,58){\makebox(0,0)[t]{Value}}
\put(65,52){\makebox(0,0)[tl]{%
  \begin{minipage}{14mm}
  \CVAL \end{minipage}}}
\put(0,68){\line(1,0){80}}
\put(14,61){\makebox(0,0)[bl]{\DBX\ Documents}}
\put(14,66){\makebox(0,0)[tl]{\GBX\ Gift}}
\put(34,61){\makebox(0,0)[bl]{\OBX\ Other}}
\put(34,66){\makebox(0,0)[tl]{\CBX\ Commercial sample}}
\put(0,80){\line(1,0){80}}
\put(2,79){\makebox(0,0)[tl]{%
  \begin{minipage}{76mm}\normalsize
```

```
  \textbf{Customs Declaration} ...
  officially  \end{minipage}}}
%% vertikální linky
\put(48,26){\line(0,1){34}}
\put(64,26){\line(0,1){34}}
\thinlines
\put(56,26){\line(0,1){9}}
\put(56,39){\line(0,1){14}}
\end{picture}
\setlength{\unitlength}{1pt}
}% konec \form
```

Názvy maker pro části, které se vyplňují, jsou velkými písmeny. Pro vyplnění formuláře je třeba tato makra nadefinovat. V uvedeném příkladu jsou definice pro vyplnění a zobrazení tyto.

```
\let\GBX\xbox    \let\DBX\tickbox
\let\OBX\tickbox \let\CBX\tickbox
\newcommand{\CONT}{\normalsize\rmfamily Toy \\ Scarf}
\newcommand{\CVAL}{\normalsize\rmfamily\centering 15 \\ 12}
\begin{figure}
\centering
\form
\end{figure}
```

4. Dopisy

Od Michaela Barra jsem obdržel následující dopis týkající se porovnávání řetězců. Možná někdo pomůže.

Po přečtení Vašeho článku v *TUGboat*u [1] jsem narazil na problém, který je dle mého názoru neřešitelný. Alespoň pro mě neřešitelný. Předpokládejme, že chceme testovat, zda je argument po úplné expanzi prázdný. Nebo zda mají dva argumenty stejnou úplnou expanzi. Nakonec jsem došel k makru podobnému jako Vaše makro `\srovnejretezretez`

```
\newif\ifstejne
\newcommand{\srovnejretezretez}[2]{%
  \stejnefalse
  \begingroup
    \def\1{#1}\def\2{#2}%
    \ifx\1\2\endgroup \stejnetrue
```

```
95      \else   \endgroup
96      \fi}
```

přičemž jsem místo `\def` použil `\edef`. Toto makro fungovalo až do té doby, než jako argument byla matice. Dostal jsem chybové hlášení o nesprávně umístěném &. Ukazuje se, že zatímco můžete matici vložit do `\def`, nemůžete ji vložit do `\edef`. Toto je vlastnost (vyslov „bug“) TEXu, která nebude opravena a kterou není možné obejít.

Michael Barr

Seznam literatury

[1] Wilson, Peter. Glisterings. *TUGboat*, 22(4):339–341, December 2001.

Summary: It might work II. – Forms

This paper shows pieces of LaTeX code that may help typesetting forms. The author presents simple macros for typesetting tickboxes and rules and he typesets the whole form using the picture environment.

Keywords: LaTeX, typesetting of forms, picture environment

Peter Wilson, herries.press@earthlink.net
18912 8th Ave. SW
Normandy Park, WA 98166 USA

Blahtexml and multi-target document generation

Gilles Van Assche

Abstract:

Blahtex and Blahtexml are open-source tools for converting mathematical expressions written in the TEX syntax into MathML. This article focuses on a particular use case, where the source of a scientific document is written in XML and can be the input for a variety of output formats, ranging from LATEX articles to documents in OpenDocument format to web pages. We show that Blahtexml can play a central role in such a context, where the author wishes to enter equations in the TEX syntax and yet enable his document for publication not only with TEX but also in MathML-based formats.

Key words: Blahtex, Blahtexml, MathML, TEX, LATEX, conversion, publishing of mathematical documents

Typing a mathematical expression using the syntax of TEX is much more convenient than in the MathML syntax. In fact, the latter was not designed to be typed by hand, but instead to be entered in a MathML editor or converted from another format. Yet, MathML is being adopted by an increasing number of programs and utilities, especially in browsers to display pages with formulas on the Web. To be able to use MathML while retaining the convenience of the TEX syntax, Blahtex(ml) provide a way to convert mathematical formulas from the syntax of TEX (or a large subset of it) to MathML [8].

Blahtexml differs from Blahtex in that it adds the ability to convert all TEX formulas in an XML file to MathML. The idea behind this new functionality stems from a specific use case of Blahtexml: the generation of documents in multiple formats from a single source [14]. This article focuses on a particular use case, where a document is written in XML and becomes the source for a variety of output formats, ranging from LATEX articles to documents in OpenDocument format to web pages. This approach is not new—actually, it is a fairly natural one—yet this article points out that Blahtexml fits nicely in the picture when it comes to scientific documents and papers.

The rest of the paper is organized as follows. First, we give an overview of XML technologies for scientific documents. Then, we describe the functionality of Blahtex. Finally, we present the single-source approach for scientific documents, including information on Blahtexml, XSLT and an example.

doi: 10.5300/2012-3/137

XML technologies

The Extensible Markup Language, or XML, has become a popular way to express the content and structure of a document [3]. XML defines a generic syntax for enriching texts (or data) with humanly-readable tags. Alone, XML is hollow—it does not define the meaning of tags, nor how to process an XML document. Instead, it can be viewed as a common ground for applications that share a single syntax and a lot of standard tools to generate, query, transform and edit data or documents in a unified way. For instance, the Extensible Stylesheet Language Transformations (XSLT) language is an efficient way to generate XML documents or to transform one XML file into another [4].

An XML application is a restriction of the XML syntax to a well-defined set of tags and other conventions. Anyone is free to define his/her own XML application. As of interest for scientific documents, there are at least three XML applications that are important to mention: XHTML, the OpenDocument format and MathML. First, XHTML is an XML version of the famous HyperText Markup Language (HTML) that describes the content of a web page [5]. Retro-compatible with HTML, XHTML is a clean version of HTML that follows the XML syntax and consequently allows to use all the XML tools. Second, the OpenDocument format uses a container format (as a Zip file) that embeds XML files for the content and style information of the document [2]. Finally, MathML is an XML application that describes mathematical expressions [1]. It encodes the structure of such expressions in a standard way, so that software can display or process them.

MathML is used for embedded formulas in several applications, including XHTML and OpenDocument. For instance, MathML formulas can be included in XHTML web pages. Traditionally, mathematical expressions have been included as bitmap pictures—this is a solution with many drawbacks (e.g., poor, non-scalable display quality, increased load time), but of course one that works for all browsers. Formulas in MathML, on the contrary, provides a better alternative, which is supported by an increasing number of software, including many recent browsers (e.g., Firefox [6], Design Science's MathPlayer plug-in [7] for Microsoft Internet Explorer).

Blahtex

While MathML is becoming a universal way to express and exchange mathematical expressions, its syntax is extremely verbose, preventing the most courageous user from entering an equation of reasonable size by hand in a text editor. In fact, it is not the purpose of MathML for one to be able to actually type a formula in this syntax. Instead, there are interactive editors or converters to do so.

Unlike MathML, the syntax of mathematical expressions in TEX is the de-facto standard in the scientific community and is simple enough to be entered by hand. This is where BLAHTEX comes into play: It allows one to enter formulas using the syntax of TEX and to convert them into MathML.

BLAHTEX was written by David Harvey, who targeted his program to support equations in MediaWiki, the engine behind Wikipedia [12]. In this context, writers enter text in a rather simple syntax called *wiki text* and MediaWiki generates the HTML code to be displayed in a browser. To keep the syntax simple, writers are allowed to enter equations in the TEX syntax. Currently, texvc converts the mathematical formulas of Wikipedia to either HTML or PNG bitmaps [11]. As an alternative, a MediaWiki extension using BLAHTEX is able to convert each of these into MathML [13]. Like texvc, BLAHTEX processes each equation individually.

The syntax supported by BLAHTEX is a subset of the TEX syntax, but the chosen subset is large enough for most purposes. For instance, it supports a long list of symbols, commands and environments compatible with TEX, LATEX and AMS-LATEX, as well as macros via `\newcommand`. The complete list can be found in the user manual [8].

Internally, BLAHTEX processes everything as Unicode, from the Greek letters to mathematical operators to text in languages other than English. As a convenient extension to the TEX syntax, BLAHTEX accepts a number of mathematical symbols to be directly entered in Unicode as an alias to the TEX command. E.g., BLAHTEX makes no difference between the multiplication sign "×" entered as is and the `\times` command.

A nice thing about BLAHTEX is that it makes a good attempt at providing the same spacing between operators as TEX does. It determines the proper spacing and provides it in the generated MathML code as `lspace` and `rspace` attributes. Although the rendering of MathML varies from browser to browser, this helps getting a consistent look, as close as possible to TEX's appearance.

We now illustrate the use of BLAHTEX through some examples.

The first way to use BLAHTEX, with the `--mathml` option, is to convert an equation given at standard input into MathML at standard output. For instance, typing: `echo '\sqrt{x^2+\alpha}' | blahtex --mathml` produces the output in Figure 1. In this example, the MathML fragment is enclosed in `blahtex/mathml/markup`. Note that the MathML fragment produced does not contain any namespace information; ideally, the MathML namespace should be added when enclosing this fragment in an actual XML file. In the case of a syntax error, explicit information is given in a `blahtex/error` element.

The second way to use BLAHTEX, with the `--png` option, is to convert an equation into a PNG file. BLAHTEX calls TEX to produce this bitmap picture. The name of the output file is automatically generated from the MD5 digest of the TEX code. Hence, if the same formula appears several times, only one PNG file is produced. To be able to determine the name of the PNG file, the digest is provided

```
<blahtex>
<mathml>
<markup>
<msqrt>
    <msup>
        <mi>x</mi>
        <mn>2</mn>
    </msup>
    <mo lspace="0.222em" rspace="0.222em">+</mo>
    <mi>&#x3B1;</mi>
</msqrt>
</markup>
</mathml>
</blahtex>
```

Figure 1: Sample MathML output provided by Blahtex

```
<blahtex>
<png>
<md5>068bd5f892d1f87b0371fa570af10712</md5>
</png>
</blahtex>
```

Figure 2: Sample PNG file name output

in the `blahtex/png/md5` element of the XML fragment at the standard output. For instance, typing `echo '\sqrt{x^2+\alpha}' | blahtex --png` produces the file `068bd5f892d1f87b0371fa570af10712.png` displaying $\sqrt{x^2+\alpha}$ and the XML fragment of Figure 2.

Single-source approach for scientific documents

When writing a scientific document, the writer wishes to concentrate on the content and not worry too much about the technical details of the typesetting system. The purpose of LaTeX, as a layer on top of TeX, is indeed to provide separation between content and presentation. However, it does not forbid the writer to enter specific commands to control details of some presentation aspects, as the need naturally arises in practice. Also, one often has a predetermined target in mind for a document (e.g., an article for a specific journal, a report, a thesis) when writing it. Having specific presentation requirements (e.g., the journal's layout) is not a problem for a single document. However, if one wishes

```
<?xml version="1.0"?>
<equations xmlns:b="http://gva.noekeon.org/blahtexml">
  <equation b:inline="x+y"/>
  <equation b:block="\exp(-\gamma x)"/>
</equations>
```

Figure 3: Sample input file for Blahtexml

to re-use material between various documents, a simple copy & paste may not be enough: Some presentation-oriented commands need to be adapted as the layout conventions for different targets may not be identical. For instance, different LaTeX class files may have slightly different syntaxes. To enter the abstract of an article, one may require to enclose it in a `\abstract` command, while others require it in an environment delimited by `\begin{abstract}` and `\end{abstract}`. As another example, the highest heading level of an article is `\section`, while it is `\chapter` for a report. Moving a section to another document or to another level may require adapting all the heading commands.

Presentation-oriented commands may become even more problematic when the output format is not LaTeX's original target (i.e., a Postscript or PDF file) but, say, a web page. It would be excessive for a converter from LaTeX to HTML to support all the presentation-oriented aspects of the document. At least some of them do not make sense at all, such as the page format, whilst others might just be very difficult to convert.

While there is no miracle solution to these problems, we think that the best solution is to generate different output formats from a source file in a common syntax. The common syntax may or may not be related to one of the output formats. The point is, however, that the common syntax should focus on the content and that, if necessary, some common presentation aspects can be added to it, provided that it does not privilege or exclude one of the output formats specifically.

Using Blahtexml

The idea of a common syntax naturally extends to the mathematical expressions, which can then be converted into an appropriate set of formats, depending on the target output format. This is where Blahtexml comes into play. Assuming a document written in a syntax based on XML, Blahtexml converts each equation found in the document into either MathML, nominal TeX syntax, PNG bitmap image files, or all three formats. The syntax of Blahtex(ml) is indeed TeX-oriented. Yet, the subset supported by Blahtex(ml) excludes TeX-specific presentation aspects that could not be converted into MathML.

```
<?xml version="1.0" encoding="UTF-8"?>
<equations xmlns:b="http://gva.noekeon.org/blahtexml">
  <equation>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <mi>x</mi>
      <mo lspace="0.222em" rspace="0.222em">+</mo>
      <mi>y</mi>
    </math>
  </equation>
  <equation>
    <math xmlns="http://[...]MathML" display="block">
      <mi>exp</mi>
      <mo lspace="0" rspace="0" stretchy="false">(</mo>
      <mo lspace="0" rspace="0">-</mo>
      <mi>&#x3B3;</mi>
      <mspace width="0"></mspace>
      <mi>x</mi>
      <mo lspace="0" rspace="0" stretchy="false">)</mo>
    </math>
  </equation>
</equations>
```

Figure 4: The output file given by BLAHTEXML for the input file in Figure 3

BLAHTEXML provides the `--xmlin` option, which does not exist in BLAHTEX. With this option, BLAHTEXML processes an input file given at standard input. Such an input file may look like the example of Figure 3. The equations are given as attributes (`inline` or `block`) in the BLAHTEXML namespace. Whenever BLAHTEXML meets such an equation, it expands it into the equivalent MathML code. The corresponding output is given in Figure 4. Note that by using a namespace, attributes containing equations can be added to any XML file independently of the syntax of other applications.

In addition to the MathML representation of the equations, the `--annotate-TeX` and `--annotate-PNG` options cause BLAHTEXML to produces an `annotation` element with the equation in nominal TEX syntax and another `annotation` element with the name of the PNG file containing a bitmap rendering. The generated MathML code and both new elements are enclosed in a `semantics` element, to conform to the MathML syntax. From the same example as above, this would generate the output of Figure 5.

```
<equations xmlns:b="http://gva.noekeon.org/blahtexml">
  <equation>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <semantics>
        <mrow>[...]</mrow>
        <annotation encoding="TeX">x + y</annotation>
        <annotation encoding="image-file-PNG">
            ./f05c46190061a618fd432bf5471cc2ab.png</annotation>
      </semantics>
    </math>
  </equation>
  <equation>
    <math xmlns="http://[...]MathML" display="block">
      <semantics>
        <mrow>[...]</mrow>
        <annotation encoding="TeX">
            \exp ( - \gamma x )</annotation>
        <annotation encoding="image-file-PNG">
            ./df6bfcabef19b8a0ccdbd2077ae96e75.png</annotation>
      </semantics>
    </math>
  </equation>
</equations>
```

Figure 5: The output file given by BLAHTEXML for the input file in Figure 3 when additional annotations are requested

Using XSLT

In document generation from a source in a common syntax, the source file of a document must be parsed before contents in the target format can be generated. Restricting the common syntax to an XML application, parsing XML can be done with various tools or can be programmed in different languages with appropriate libraries. Among the available tools, the XSLT language is particularly well suited for transforming an XML source file into either another XML file or a text file. Let us briefly introduce this tool and explain why it is well suited to our particular use case.

In XSLT, a *stylesheet* defines a transformation from XML into either XML or text. In its simplest form, it is a declarative language that specifies the piece of text or XML data to generate when it encounters a given XML tag in the source file. To apply a given stylesheet to a source file, one uses an *XSLT processor*.

```
<xsl:template match="b">
    <xsl:text>\textbf{</xsl:text>
    <xsl:apply-templates/>
    </xsl:text>}</xsl:text>
</xsl:template>
```

Figure 6: Example of XSLT code to convert the bold `b` tag of XHTML to the `textbf` command in LaTeX

XSLT can become a bit complex when the task to perform diverges from its core abilities. However, in the context of multi-target document generation, XSLT is simple to program and to read. For instance, no explicit loops need to be written to go through the entire source file, as such loops are managed by the XSLT processor automatically. This reduces the work to writing the text or XML fragment to be generated corresponding to a given input XML element.

As a brief example, let us consider the conversion from XHTML to LaTeX using XSLT. The XHTML tag `b` indicates bold text. The equivalent LaTeX command would be `\textbf`. The piece of code in Figure 6 makes this conversion: It declares a template, which matches `b` tags. For all such tags, it then tells to output `\textbf{`, then to apply recursively other templates, e.g., to convert other tags or simply to write the text inside the `b` tag, and finally it concludes by outputting the closing brace `}`.

A simple example based on XML

On the BlahTeXML web page, we provide an example of document generation system based on an XML syntax [10]. This is a working example, although with a reasonably simple functionality. The goal is not to rival with well-known systems, such as DocBook [9], with its definition of almost 400 different tags. Instead, this working example proposes a clean and simple syntax, whose only ambition is to illustrate the use of BlahTeXML for multi-target document generation in the scope of scientific documents and articles.

The proposed example is based only on open-source technologies: The general process is managed by `make` and the XSLT processing is performed by any XSLT processor. In the example, the processor used is `xsltproc` [15], although any XSLT processor could be used.

Let us briefly describe the syntax of the source file and then the process from the source file to a target output. The source file is a document in XML, which contains the text, the structure of the document and some meta-information. The input syntax is illustrated in the file `Sample.ed`, which contains some sample text and mathematical expressions. The root element of the XML file is `document`. In it, two child elements appear: `head` and `body`. In the former, information about

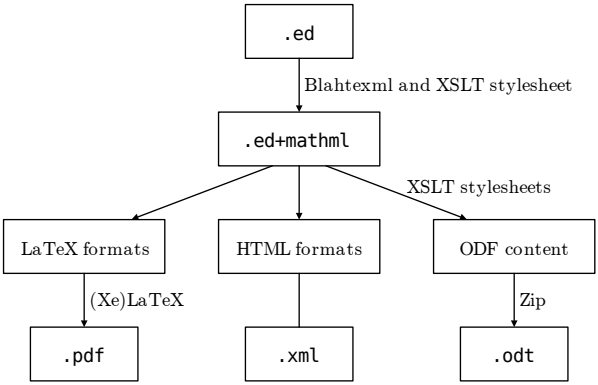


Figure 7: The general flow

the author(s), their affiliation and the title can be provided. The latter provides the contents and structure of the document.

The structure of the file was inspired from XHTML 2.0. Text paragraphs can be grouped in sections using the `section` element. Such sections can be nested, which mean they actually represent a chapter, a section or a subsection depending on the nesting depth. Section titles are provided in `h` elements. Text paragraphs are enclosed in `p` elements, and ordered and unordered lists in `ol` and `ul`, respectively, with each list item in `li`. Inside paragraphs or list items, plain text can be given. The text can be further formatted using emphasis (italic, `em`), a strong font (bold, `strong`), small capitals (`sc`), subscript (`sub`) and superscript (`sup`).

As of interest for BLAHTEXML specifically, inline mathematical expressions are written in `ieq` elements, and stand-alone formulas in `eq` elements. Inside such elements, the formula is given in BLAHTEX format.

The general processing flow is illustrate in Figure 7. To determine the sequence of steps from the source file to the output file, a `makefile` is provided. Depending on the target format, the following steps can be taken:

- As a the mathematical expressions are not written as attributes (but more simply inside `ieq` and `eq` elements), a first step consists in putting the equations in the appropriate attributes for BLAHTEXML. This preparation step is performed by the `PrepareForBlahtexml.xsl` XSLT stylesheet.
- As a result of the previous step, the mathematical expressions are written as attributes in the BLAHTEXML namespace. This step now consists in

converting these formulas using BLAHTEXML with the `--annotate-TeX` and `--annotate-PNG` options. As a result, all formulas are in three formats: MathML, TEX and as PNG files. Depending on the desired output format, the following steps will extract the format they need.

- Then, the XSLT stylesheets `Numbering.xsl` and `Referencing.xsl` process the resulting file to number sections and to resolve cross-references. This step is mainly done for XHTML output, as LATEX and OpenDocument formats have their own syntax to express numbered sections and references.
- The core of the output generation is performed by a format-specific XSLT stylesheet to produce XHTML, LATEX or OpenDocument code. More details on the various output formats are given below.
- Optionally, a last step finalizes the production and again depends on the desired output format. For instance, for a `.tex` file, LATEX (or XELATEX) is invoked to produce a PDF file. If the target format is OpenDocument, then the resulting XML file is packaged into a Zip file and renamed as `.odt`.

Let us give some more details about the generation of the possible output formats. To allow `make` to determine which sequence of operations to perform, the different output formats have specific extensions. For instance, to produce a PDF file from `Sample.ed` via LATEX using an IEEE class file, one has to type `make Sample.ieee.latex.pdf`. We will see the other extensions as we go.

For XHTML, the generation of the various tags is fairly straightforward, since to an element of our input syntax corresponds an element in XHTML. This part of the job is done by the `ToXHTML-common.xsl` stylesheet. Details about the display styles can be tuned via the `document.css` cascaded stylesheet. There are three flavors of XHTML output formats, depending on the way the mathematical expressions are handled.

- For equations in MathML, the extension is `.xhtmlmathml.xml` (e.g., `make Sample.xhtmlmathml.xml`).
- As a first alternate option for browsers that have no MathML support, the mathematical expressions can be displayed as bitmap pictures, using the PNG files produced earlier. For this, the extension is `.xhtmlpng.xml`.
- As a second alternate option, the mathematical expressions can be displayed with pure HTML tags, but in a rather approximate form. For instance, HTML can display text in superscript and subscripts, but if an expression (like $A_{\text{sub}}^{\text{sup}}$) requires both then the HTML code will not be able to put one above the other (e.g., the result might look like $A_{\text{sub}}{}^{\text{sup}}$). Other restrictions apply, for instance, for two-dimensional constructions such as matrices or fractions. Nevertheless, this option may be useful and sufficient if the formulas are simple. Here, the extension is `.xhtmlapprox.xml`.

For TEX and derivatives, there are also several flavors. In the provided example, the output is either LATEX-oriented or XELATEX-oriented. The latter has the advantage of an easy support of True Type and Open Type fonts. Here the XSLT stylesheet must output a text file that follows TEX's syntax conventions. The main part of the job is done by the `ToLaTeX-common.xsl` stylesheet. Then, a number of smaller XSLT stylesheets give specific generation rules, most notably a specific header, for each flavor. The flavors supported in this example are the following.

- For a simple article in LATEX, the specific stylesheet is `ToLaTeX-article.xsl` and the extension is `.article.latex.tex` (e.g., `make Sample.article.latex.tex`) for the `.tex` source file. To get the result directly as a PDF file, the `.tex` extension can be replaced by `.pdf` (e.g., `make Sample.article.latex.pdf`).
- For an article following the APS Physical Review conventions and using the `revtex4-1` class file, the stylesheet is `ToLaTeX-revtex.xsl` and the extension is `.revtex.latex.tex`.
- For an article using the `IEEEtran` class file for the IEEE Transactions journals, the stylesheet is `ToLaTeX-ieee.xsl` and the extension is `.ieee.latex.tex`.
- For a simple article in XELATEX, the stylesheet is `ToXeLaTeX-article.xsl` and the extension is `.article.xelatex.tex`.

Adding a new flavor tailored to special needs is rather simple, as it suffices to add a new rule in the `makefile` and a new XSLT stylesheet based on one of the models above. Most of the specific stylesheets just define an alternate LATEX file header.

Finally, for OpenDocument format, most of the job is done by the `ToODT.xsl` stylesheet. It produces a file called `content.xml`, which is then Zipped, together with the files provided in `ODT-Template/`, to make a `.odt` file. Here the target extension is simply `.odt` (e.g., `make Sample.odt`). Details about the display styles can be tuned in the `ODT-Template/styles.xml` file. The `.odt` file can be opened by any word processor supporting the standard OpenDocument format[1].

[1] At this time of writing, a bug in OpenOffice.org prevents the mathematical expressions from being displayed with a correct size after loading the document [16]. A possible workaround consists in double-clicking on the equations to open them in the integrated equation editor, which forces OpenOffice.org to resize the mathematical expressions appropriately. We hope this issue will be solved soon.

Conclusions

BLAHTEX(ML) can be useful for converting mathematical expressions written in the TEX syntax into MathML. In particular, we have shown that BLAHTEXML can perform this task in the scope of a multi-target document generation system for scientific documents. We have given an example to illustrate this purpose, where a document is written in a common XML-based syntax and various output formats can be generated from it, including various flavors of LATEX.

Although fully working, the example given is rather simple from a functionality point of view. In this respect, future work may include the support for tables, figures, bibliographic entries and more output formats.

References

[1] WORLD WIDE WEB CONSORTIUM. *Mathematical Markup Language (MathML)*. http://www.w3.org/standards/webdesign/math.

[2] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS. *Open Document Format for Office Applications (OpenDocument)*. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office.

[3] WORLD WIDE WEB CONSORTIUM. *Extensible Markup Language (XML)*. http://www.w3.org/standards/xml/.

[4] WORLD WIDE WEB CONSORTIUM. *Extensible Stylesheet Language Transformation (XSLT)*. http://www.w3.org/standards/xml/transformation.

[5] WORLD WIDE WEB CONSORTIUM. *HTML & CSS*. http://www.w3.org/standards/webdesign/htmlcss.

[6] MOZILLA. *Firefox*. http://www.firefox.com.

[7] DESIGN SCIENCE. *MathPlayer plug-in*. http://www.dessci.com/en/products/mathplayer/.

[8] HARVEY, D., ASSCHE, G. VAN. *Blahtex and blahtexml version 0.8 manual*. http://gva.noekeon.org/blahtexml/.

[9] WALSH, N. *DocBook 5: The Definitive Guide*. O'Reilly, 2010.

[10] ASSCHE, G. VAN. *ExampleDoc*. http://gva.noekeon.org/blahtexml/exampledoc.

[11] WEGRZANOWSKI, T. *Texvc*. http://en.wikipedia.org/wiki/Texvc.

[12] WIKIMEDIA FOUNDATION. *MediaWiki*. http://www.mediawiki.org/.

[13] MEDIAWIKI. *Extension: Blahtex*. http://www.mediawiki.org/wiki/Extension:Blahtex.

[14] WIKIPEDIA. *Single source publishing*. http://en.wikipedia.org/wiki/Single_source_publishing.

[15] VEILLARD, D. *The xsltproc tool*. http://xmlsoft.org/XSLT/xsltproc2.html.

[16] OPENOFFICE.ORG. *Issue 91779*. http://www.openoffice.org/issues/show_bug.cgi?id=91779.

Souhrn: Blahtexml and generování dokumentů v různých formátech

Blahtex and Blahtexml jsou nástroje typu „open-source“ pro koverzi matematických výrazů zapsaných syntaxí jazyka TEX do MathML. Tento článek se zaměřuje na konkrétní příklad, kde zdroj vědeckého dokumentu je zapsán v XML a může být vstupním formátem pro konverzi do celé řady formátů výstupních, od článků psaných v LATEXu přes formát OpenDocument až po webové stránky. Ukážeme jak Blahtexml může hrát významnou roli, když autor si přeje vkládat rovnice v syntaxi TEXu, ale současně chce umožnit publikování svého dokumentu ve formátech odvozených z MathML.

Klíčová slova: Blahtex, Blahtexml, MathML, TEX, LATEX, konverze, publikování matematických dokumentů

LaTeX at Distributed Proofreaders and the Electronic Preservation of Mathematical Literature at Project Gutenberg

Andrew D. Hwang

Abstract:

A small but growing effort is underway at the volunteer web site Distributed Proofreaders (DP, at `www.pgdp.net`), with the goal of creating high-quality LaTeX files of selected public domain mathematical books for distribution by Project Gutenberg (PG). This article introduces DP and PG, describes how books are transcribed at DP, and gives an overview of current LaTeX coding strategies.

Keywords: LaTeX, Distributed Profreading, Project Gutenberg, Proofing, Formatting, Post-processing

Introduction

Public domain mathematical works are a precious resource. Electronic preservation potentially makes historical mathematical literature available to anyone with a computer. By contrast, printed books and journals stored in university libraries suffer from constraints ranging from limited access to physical degradation.

This article describes a small but growing initiative to harness "crowdsourcing" for the purpose of transcribing public domain mathematical works into LaTeX. The existing web-based infrastructure is provided by Distributed Proofreaders (DP, at `www.pgdp.net`). The completed books are distributed by Project Gutenberg (PG, at `www.gutenberg.org`). The LaTeX work at DP and the availability of LaTeX source files for mathematical projects at PG are not widely-known. Please share this article with interested students and colleagues, and explore the sites yourself.

Since 2008, more than fifty LaTeX books have been produced at DP [1]. Recently-completed examples range in subject matter and sophistication from popular accounts to textbooks to research monographs. Titles include:

- `http://www.gutenberg.org/etext/26839` *Mathematical Recreations and Essays* by W. W. Rouse Ball,
- `http://www.gutenberg.org/etext/28233` *Philosophiae Naturalis Principia Mathematica* by Sir Isaac Newton,
- `http://www.gutenberg.org/etext/31246` *A Short Account of the History of Mathematics* by W. W. Rouse Ball,

- `http://www.gutenberg.org/etext/31564` *Vorlesungen über Thermodynamik* by Max Planck,
- `http://www.gutenberg.org/etext/33283` *Calculus Made Easy* by Sylvanus P. Thompson.

The medium-term goals for LaTeX book production at DP are twofold: First, to recruit and build a community of LaTeX-knowledgeable volunteers; and second, to select and prepare suitable books from the mathematical literature of the 19th and early 20th Centuries. Further, DP can process any book for which copyright clearance is obtainable. Authors willing and able to grant perpetual, non-exclusive, worldwide rights to distribute their books in electronic form on a royalty-free basis can, at no cost to themselves, have their books converted to electronic form and made available at PG. A self-sustaining LaTeX community at DP stands equipped to generate a lasting scientific, cultural, historical, and educational resource.

Techniques of ebook production

Broadly speaking, "electronic preservation" may refer to anything from scanning a book and distributing bitmap image files (jpegs or pngs) to preparing an accurate, archival-quality textual representation, such as a well-designed LaTeX source file.

Scanning a book is relatively cheap and fast. A book of a few hundred pages can be scanned manually and non-destructively in about an hour by one individual without special skills or expensive equipment. Books can also be scanned destructively in bulk at high speed by cutting off the spine and running the pages through a mechanical feeder. At this writing and for the foreseeable future, the vast majority of mathematical ebooks consist of bulk-scanned images.

Once a book has been scanned, raw text may be extracted fairly easily with optical character recognition (OCR) software. Not surprisingly, however, mathematics is rendered poorly by OCR. As a result, raw OCR text of a mathematical book is all but unusable for a casual reader.

At DP, OCR text is the input material. Human volunteers carefully proofread the text against the page scans, then add LaTeX markup. The end result is an accurate textual and semantic representation of the book. Though producing a high-quality LaTeX source file requires on the order of an hour of skilled work *per page*, the benefits are substantial. For the typical reader, a LaTeX-produced PDF file is text-searchable, magnifiable on screen without loss of quality, easily--hyperlinked, and yields camera-quality printed output. To the benefit of readers without fast Internet access, a LaTeX-produced PDF file is about one-tenth the size of a collection of page scans; a compressed source file is smaller still. Thousands of textual books can be fit onto a DVD, compared with a couple hundred books made from scanned images. A good-sized library can therefore be

easily and inexpensively distributed worldwide by ordinary post. Finally, if the coding is well-planned, a LaTeX source file can serve as an archival representation of the book.

Project Gutenberg and Distributed Proofreaders

Founded by Michael Hart at the University of Illinois in 1971, Project Gutenberg is the world's oldest electronic library. PG is dedicated to the storage and distribution of public domain ebooks.

Distributed Proofreaders was founded in 2000 by Charles Franks to produce ebooks for PG. The site source code, written in PHP, is free software released under the GNU GPL. The project homepage is `dproofreaders.sourceforge.net`. At this writing, there are at least six independent "DP sites" using some version of the code base. In addition to the DP site at `www.pgdp.net`, there are smaller "sister" DP sites based in Canada and Europe, which operate under the copyright laws of their respective regions. Due to lack of infrastructure and volunteers, LaTeX projects are currently processed only at `www.pgdp.net`, and the present article describes only activities at this DP site.

DP currently boasts a few hundred volunteers active on a near-daily basis, and produces a little over half of the new ebooks in PG's collection. At this writing, the number of volunteers who work on LaTeX is about 1% of the "population", and on average about 20 new LaTeX books are posted to PG every year.

The DP site at `www.pgdp.net` was designed and built entirely by volunteers, and is currently staffed by volunteers. DP-Canada, DP-Europe, and Project Gutenberg are also largely or entirely built and run by volunteers.

DP *process overview*

An ebook starts its life at DP as raw OCR output. The page-length pieces of OCR text and the page scans are loaded into a database hosted at DP. Working one page at a time, volunteers at the DP web site are presented with a scanned page image side-by-side with the corresponding OCRed text in an editor window. After correcting the text and adding LaTeX macros, proofreaders check the page back into the database. Once all the pages of a book have been reviewed and corrected, the site software concatenates the pages into a raw ebook file. A single volunteer performs final polishing and verification, then submits the completed ebook to Project Gutenberg.

The actual path of a book through DP is a bit more involved. The distributed work is separated into "proofing" and "formatting" stages. Proofing focuses on verifying the correctness of the raw words in the text, the general dictum being "match the scan". Because most DP volunteers do not speak LaTeX, the text file at the end of the proofing rounds omits most of the mathematical markup, and

is far from being machine compilable. The formatting rounds add the necessary markup, including mathematics, footnotes, and sectional divisions. The output of the formatting rounds is, with minor modifications, machine compilable once the appropriate preamble has been prepended, but is still far from a completed ebook. The remaining work on a project, generically termed "post-processing" and typically comprising about 25–35% of the total production time, is performed off-line.

Coding for longevity

Data formats are a troublesome fact of life for long-term electronic encoding and storage of information. Electronic documents become useless when there is no easy, reliable way to recover the textual and presentational information stored in a file format.

Storage in an open, non-proprietary, plain text format guards against lossage due to lack of decoding software. The textual content of a LaTeX source file will remain accessible as long as computers can read plain text in a present-day encoding. However, LaTeX markup alone does not guarantee longevity; far from it. Used as a WYSIWYG tool, even the most capable markup language cannot capture more than a book's visual appearance.

For longevity, flexibility, and ease of maintenance, a source file needs to separate four interrelated but distinct aspects: (i) textual content (maintaining readability by both people and machines), (ii) semantic structure, (iii) visual presentation and layout, and (iv) implementation in terms of typesetting software.

Carefully-planned macros meet all four requirements, embodying these multiple layers of structure, both clarifying the code and simplifying the task of future maintainers who wish to convert today's LaTeX files into source files suitable for the typesetting software of 2050 and beyond. Technical details of DP's current practices are surveyed in Section below.

The structure of DP

Since the production of mathematical ebooks at DP takes place within an infrastructure designed primarily for HTML-formatted projects, it is worth describing the general organization and operation of DP in parallel with the special requirements and practices of the LaTeX community.

DP is primarily an English-language site. For LaTeX projects, English-language books are generally preferred, though a number of books in French and German have also been produced. The site code currently restricts source files to the Latin-1 (`iso-8859-1`) encoding, so a book's language must be representable in Latin-1. (DP-Canada and DP-Europe can handle `utf-8` with some limitations.)

There are four major phases of ebook production at DP: content providing, proofing, formatting, and post-processing. Each has its own time commitments, skill set, and access requirements [2].

Content providing

A content provider (CP) conveys a collection of page scans, possibly including OCR output, to an experienced DP volunteer known as a "project manager". Scans may be "harvested" from a third party such as the Internet Archive, or may be scanned by the CP. A "copyright clearance" must be obtained from Project Gutenberg before scans are uploaded to DP [4].

If you would like to have a specific work transcribed at DP, please contact the author of this article or post in the "LaTeX Typesetters Team" in the DP forums.

Selecting suitable books

Books should normally be selected primarily for expected popularity or value as scholarly references. A new LaTeX project should not be initiated unless a volunteer expresses the *commitment* to post-process.

Given the current size of the LaTeX community at DP, the best books are in the vicinity of 250 pages or fewer, and contain mostly uniform, straightforward typography, and only mathematics that can be easily typeset using the AMS math environments.

Books should generally be avoided if they contain extensive typography that is relatively difficult to render in LaTeX, such as long division, tabular data with many multi-row or multi-column alignments, multi-column lists of exercises and answers, typography that changes at each paragraph (as in a geometry textbook), or large numbers of illustrations, particularly inset diagrams.

Proofing

The "distributed" portion of ebook production at DP has well-developed guidelines designed to allow most pages of most books to be processed uniformly. When questions arise of how to handle unusual constructs, volunteers may communicate with each other and with the project manager via phpBB bulletin boards. Each project has a dedicated discussion thread. There are also dozens of forums for general questions.

Normally, each page of a book passes through three rounds of proofing, named P1–P3, with successive passes made by volunteers having greater experience and ability at catching errors. Once all pages of a project have completed a round, the project is made available in the next round. At any given time, a project is "in" a specific round, and each page of a project is proofed the same number of times.

In the proofing rounds, volunteers ensure that the characters in the text file match the characters in the page scan. In other words, the focus is on content.

In a LaTeX project, the first proofing round typically involves considerable "type-in", or manual entry of characters, because OCR handles mathematics so poorly. A single page may require 10 or 15 minutes' work, a substantial fraction of the expected total preparation time.

Formatting

After the proofing rounds, each page goes through two rounds of formatting, F1 and F2. The formatting rounds capture the book's structure: chapter and section headings, quotations, footnotes and sidenotes, tables, and figures. In LaTeX projects, mathematics is coded primarily by the formatters.

For a LaTeX project, F1 entails a similar amount of type-in to P1. Additionally, a "formatting coordinator" (see Section) provides a "working preamble" for the project. Volunteers are expected to test-compile each page before marking it as "done", and to check the compiled code visually against the page scan. This amount of work makes F1 the most time-consuming round for LaTeX, about 10–20 minutes' work per page.

Post-processing

After a project leaves the rounds, the distributed phase is complete. The remaining work is done by a volunteer playing the role of "post-processor" (PPer).

A PPer downloads the formatted concatenated text and polishes it into an ebook, regularizing and finalizing the LaTeX code. Normally, a PPer becomes involved with a project before the project reaches the formatting rounds and serves as the formatting coordinator, ensuring the project is formatted according to the PPer's wishes.

PPing is complex and time-consuming, requiring fairly extensive planning and about 10–20 minutes' work per page for a modestly-complex book. At the same time, PPing provides an outlet for organizational creativity and typographical artistry, and is therefore one of the most satisfying and intellectually challenging tasks at DP.

Access requirements

Access to various activities at DP is granted according to time on site, number of pages processed, and/or peer review of one's work. Each DP site has its own set of certification requirements. Criteria for the DP site at `www.pgdp.net` are described here.

New volunteers are immediately granted access to P1. Access to P2 is granted once a volunteer has been registered for 21 days and has proofed at least 300 pages.

Certification to work in the third round of proofing is granted by application only, upon satisfactory performance under detailed human evaluation of the volunteer's proofing work. In order to apply for P3, a volunteer must have been registered at DP for at least six weeks, and have proofed at least 150 pages in P2, and formatted at least 50 pages.

F1 access is granted with access to P2. F2 certification is granted by application only, after detailed human evaluation of the volunteer's formatting work. In order to apply for F2, one must have been registered at least 91 days and have formatted at least 400 pages.

Access to PP is granted *pro forma* by request after 400 pages have been formatted. New PPers must submit their completed projects for detailed inspection by an experienced "PP Verifier" (PPVer). The PPVer assigns a "grade" to the project based on the project's length and difficulty, and the numbers of errors present in the uploaded project. After completion of eight consecutive projects with sufficiently high grade, a PPer is given "direct upload" status, and may upload projects directly to PG without supervision.

Time commitments

Volunteers at DP devote as little or as much time to the site as they like. A page is the smallest unit of proofing or formatting, and for a LaTeX project typically entails 5–20 minutes' work. Many volunteers do just one page whenever they can, perhaps every week or few. Others find the work mildly but pleasantly addictive, and work an hour or more at a sitting, several times per week.

Compared to proofing and formatting, PPing involves an extended commitment of time and energy. An experienced PPer may be able to complete a 150-page book in as little as 40 hours, but longer or more complex books can easily absorb upward of 100 hours.

Documentation and LaTeX requirements

The guidelines for proofing, formatting, and post-processing LaTeX are detailed in a set of manuals [5]. These and other LaTeX-related information applicable to DP may be found in the DP wiki [3].

DP LaTeX coding strategies

This section discusses, in some technical detail, current practices for coding LaTeX at DP. Most of these ideas are not new, but neither do they seem widely-articulated. These strategies need not be studied except by volunteers who intend to post-process, but their rationale must be consciously and continually

remembered when working at DP, where the page-at-a-time interface naturally leads a formatter to focus detrimentally on small-scale document structure.

Textual content

When a scanned page is OCRed, the output text contains the same line breaks as the printed book. Of course, the original pagination and line breaks need not and cannot be retained in a compiled PDF file. To the extent possible, however, line and page breaks *are* retained in the LaTeX source file. At DP, hyphenated words are rejoined, but otherwise there is no rewrapping of lines. Page separators are retained as LaTeX comments. The source file is therefore a reasonable visual copy of the original book, facilitating the tasks of proofreaders and eventual document maintainers.

Page and footnote numbers depend upon the document's pagination, and are not retained in the compiled output file. Other than this, textual content is retained in the document body. Particularly, LaTeX's auto-numbering is suppressed. Chapters, sections, numbered items, theorems, and equations are tagged manually, usually with the original numbering or labels represented as macro arguments. These labels have been assigned in the print version, and are *de facto* part of the original text.

Structural macros, e.g. `\Chapter`, `\Section`, `\Figure`, `\begin{Theorem}` and `\end{Theorem}`, or `\Proof`, normally generate text similar to the macro name, and do not generate more text than necessary. For example, even if most proofs begin with the phrase: "**Proof**: We must show that...", a `\Proof` macro would generate the word "Proof" in boldface, but would *not* generate the phrase "We must show that". The aim of LaTeX coding at DP is to separate content and typographical presentation in the document body and preamble, respectively. To the extent possible, the source file should be searchable for words and phrases appearing in the original book. Detailed knowledge of the preamble should not be prerequisite to reading the textual content of the book from the document body.

Semantic structure

A document body should contain few commands explicitly specifying how a piece of text is to be typeset. Instead, the body contains mostly mnemonic, high-level structural information: "this is a chapter", "this is a theorem", "this is a figure", and so forth.

The goal of semantic coding frequently turns out to be non-trivial to implement. Proofers and formatters see only one page of a book at a time. How, without inspecting a large fraction of pages, is a formatter to know the meaning of a boldface, run-in heading, or of centered italics? What if only some theorem statements are italicized; are the italics significant, or was the typesetter merely inconsistent?

At DP, a "formatting coordinator" inspects the entire book before the project leaves the proofing rounds, notes the major semantic structures and any typographical irregularities, then writes a "working preamble" for use during the formatting rounds. Ideally, the working preamble macros satisfy a number of disparate requirements. They are easy to remember, do not require formatters to type much, give a good approximation to the page scan when a formatter test-compiles a single page, and capture enough information to match the book's global typography (running heads, table of contents entries, PDF bookmarks, hyperlinks, and the like) in post-processing. For example, the text of a chapter heading might progress through the proofing and formatting rounds like this:

```
CHAPTER III: Curvature   % Proofed
\CHAPTER{III: Curvature} % Formatted
\Chapter{III}{Curvature} % Uploaded
```

All the typographical work is centralized in macro definitions.

As suggested by this code snippet, structural macros in the working preamble should *not* normally be standard LaTeX commands such as `\chapter`. Sectioning commands of LaTeX's document classes are designed with different aims than than are required at DP: They provide unwanted numbering, and are often non-trivially integrated into the document class using modern typographical assumptions. In a DP-era book, for example, a new chapter might not re-set the running head, might not start recto, and might not even begin on a new page. However, redefining the `\chapter` command accordingly also changes the behavior of the table of contents, preface, appendices, and index, probably in undesired ways.

Instead, it's preferable to add an interface layer between structural macros in the body and their implementation in terms of LaTeX commands. A `\Chapter` command in the working preamble might be implemented with the LaTeX `\section*` command. In post-processing, only the macro definition, *not the formatters' code*, needs to be modified in order to achieve the necessary typographical and cross-referencing effects.

This technique beneficially centralizes the document's dependence on the compilation engine. If typesetting software changes, only the macro definitions need modification, not every occurrence in the document body. Amplifications of this strategy are used at DP to help ensure stylistic uniformity, and to match the original typography with relative ease.

Visual presentation

DP volunteers express a wide range of opinions on how much effort should be spent making a book resemble the original, or whether ebooks should be optimized for printing (two-sided layout, generous margins) or for ebook readers (single-sided layout, very tight margins, colored hyperlinks).

There is an obvious trade-off between attractive layout on one hand and flexibility in accommodating different ebook readers on the other. This trade-off is strongly dependent on the original book; floating tables and illustrations, or even complex mathematical displays, are difficult to lay out well unless the text block size is known. As ebook readers with varying screen sizes proliferate, post--processors will encounter increasing difficulty in ensuring that finished ebooks look good on a variety of hardware.

Centralized structural coding described above facilitates the task of creating a flexible, camera-quality ebook.

Structural coding also sidesteps an issue that plagues WYSIWYG authors: Ensuring visual consistency. If section headings are printed in centered boldface type and these typographical attributes are specified explicitly for each section, the section headings are all but impossible to make identical, or to tweak and maintain.

These facts of document design are easy to see at the level of authoring an entire document, but are remarkably easy to forget when one is working one page at a time in the DP formatting rounds. The experience of years past shows that even experienced LaTeX coders incline toward hard-coding visual markup under real-life circumstances.

Implementation

In addition to the previously-noted benefits of separating structure, presentation, and content, well-planned semantic coding and encapsulating interfaces can guard against changes to external software.

A LaTeX source file obviously depends for compilability on external packages and the LaTeX kernel itself. For the LaTeX kernel and "core" packages, the need for backward compatibility helps ensure that *user interfaces* do not change. By contrast, kernel and package *internals* are all but guaranteed to be re-written beyond recognition on a time scale of decades.

On occasion in years past, LaTeX-knowledgeable post-processors at DP have concluded that a book's typography can be matched elegantly by redefining macros in a standard document class. In retrospect, this strategy is ill-advised: It relies on software internals over which the post-processor has no control.

At DP, the goals of structural markup and consistent visual presentation are achieved through factoring of typographical "responsibilities". A three-level scheme, inspired by object-oriented programming, has proven itself over dozens of projects.

Structural macros At the highest level, used in the document body, are purely structural macros needed to mark the book's semantics: `\Chapter`, `\Section`, `\Proof`, and the like.

Semantic operations In even a moderately complicated book, multiple sectioning commands need to perform identical abstract typographical operations, such as "set the running heads", "write an entry to the table of contents", "create a PDF bookmark", "include a graphic with a default width from a specified directory", or "get to a recto page, clearing the stale running head on the preceding verso page if necessary". For flexibility, visual consistency, and ease of maintenance, these operations should be factored out. Macros at this second level are not normally called directly in the document body, but only in the preamble, in the definitions of structural macros.

Depending on the book's complexity, common features among *semantic* macros may be best factored out as well. Generally, therefore, even second-level macros might *not* be implemented directly in terms of LaTeX commands.

Visual implementation The commands used to effect the visual presentation lie at the third level. These include both abstract operations such as "set the format and location of the page numbers" or "select the font of the running heads", and specific, concrete operations such as "make this text boldface". These macros, at last, are implemented in terms of standard LaTeX commands, including facilities provided by external packages.

Remarks and caveats

Abstraction and encapsulation do not always buy flexibility, and should not be used needlessly. Standard LaTeX macros, such as mathematical symbols, AMS displayed math environments, and `\footnote` commands are used routinely.

Naturally, a macro system must be designed from the top downward, based on inspection of the entire book. First determine the necessary semantic structures, then find and factor out typographical and cross-referencing operations common to two or more structural operations, and finally implement any common operations in terms of LaTeX commands.

The three layers of abstraction above are important mostly when one wishes to mimic the printed appearance of the original book. When a project warrants this level of coding, the typographical appearance can be fine-tuned easily, accurately, and consistently.

For simpler projects, this scheme may be overly elaborate. Further, if the default appearance of a standard document class is acceptable, coding semantically in terms of LaTeX's sectioning macros may be entirely workable.

Using primarily structural macros in the document body helps ensure the book will be machine-convertible to other formats, even formats not yet in existence, with as little fuss as possible. No one holds the illusion that DP's LaTeX projects can be trivially converted to other formats. However, a thoughtfully-coded ebook should be convertible to a new format with perhaps a few hours' work, compared to the dozens or hundreds of hours required to digitize the project initially.

Floats and inset illustrations Figures, tables, and complex displayed mathematics are simply a problem for current ebook readers, whose screens may be only a few inches wide.

Inset illustrations are a common cause of "brittle" documents, code whose compiled quality depends sharply on the size of the text block. The `wrapfig` package is powerful, but has relatively tight constraints on how it can place diagrams. In particular, a single paragraph cannot contain more than one `wrapfigure` environment, and mid-paragraph placement requires manual coding.

It is currently considered acceptable at DP to hard-code the placement of wrapped illustrations, but arguably it is more robust (though less pleasant typographically) to use ordinary `figure` environments instead.

DP-specific coding tricks Proofers and formatters at DP commonly make in-line notes regarding misspellings, visual obscurities, notational inconsistencies, even factual errors. Two simple macros, `\DPnote` and `\DPtypo`, are used to leave notes in the source file. `\DPnote` is a one-argument null macro. `\DPtypo` accepts two arguments, the original text and the putative correction. Changes are trivially switched on (or off) by changing one line of preamble code. Future maintainers can easily review all such changes by searching the source file for the macro name.

DP post-processors commonly use the `ifthen` package and a boolean switch to control layout suitable for printing or for an ebook reader. Again, the behavior is trivially toggled by editing one line in the source file. The scope of this technique is limited, however. Unless a book contains few or no inset diagrams, the respective print and screen layouts must, practically speaking, have the same text block size.

The future

This is potentially an exciting era for LaTeX at DP; training guidelines have been written and a stable work flow has emerged after an early period that relied on the skills of specific individuals. Whether or not DP contributes substantially to the preservation of mathematical literature in the coming years depends on its ability to build a self-sustaining community of dedicated volunteers.

Future projects should be chosen according to criteria ranging from scholarly or pedagogical value to expected popularity. Content providers must candidly evaluate a book's "value" and typographical needs, and appraise whether or not the book justifies the necessary labor to produce in LaTeX.

LaTeX-capable formatters are needed simply to distribute large amounts of work among many volunteers. What takes one formatter a month can be done by ten volunteers in a few days. Encouraging students to work at DP can both

provide valuable LaTeX coding practice and serve as an introduction to document design and planning.

For students writing a thesis, post-processing can be an avenue to working with book-length manuscripts. Naturally, PPing at DP has distinctive requirements from "ordinary" mathematical authorship, but many skills are transferable.

The contribution of just one proofed or formatted page per day from a dozen new volunteers would substantially increase DP's current LaTeX throughput. Thoughtful suggestions for new content will help ensure that important mathematical works will be available electronically for posterity.

Reference

[1] The catalog of LaTeX projects produced at DP, `http://www.pgdp.net/wiki/List_of_LaTeX_projects/Posted`.

[2] The DP wiki, `http://www.pgdp.net/wiki/Main_Page`.

[3] The DP wiki LaTeX links, `http://www.pgdp.net/wiki/LaTeX_Links`.

[4] The Project Gutenberg copyright clearance page, `http://www.gutenberg.org/wiki/Gutenberg:Copyright_How-To`.

[5] LaTeX documentation for DP, `http://mathcs.holycross.edu/~ahwang/pgdp/dptest/index.html`.

Souhrn: LaTeX na webu Distributed Proofreaders a elektronické udržování matematické literatury v projektu Gutenberg

Snaha dobrovolníků vytvářejících web Distributed Proofreaders (DP, `www.pgdp.net`) je v plném proudu. Cílem webu je vytvořit vysoce kvalitní soubory v LaTeXu vybraných volně šiřitelných matematických knih pro distribuce v rámci projektu Gutenberg (PG).

Článek představuje DP a PG a popisuje jak jsou knihy zpracovávány v DP. Dále ukazuje přehled současných kódovacích strategií LaTeXu.

Klíčová slova: LaTeX, Distributed Proofreaders, korektury, projekt Gutenberg, korektury, formátování, post-processing

Department of Math and CS
College of the Holy Cross
Worcester, MA 01610-2395, USA
ahwang (at) mathcs (dot) holycross (dot) edu
http://mathcs.holycross.edu/~ahwang/

Spolupráce TEXu se systémem počítačové algebry Sage

Robert Mařík

Abstrakt

V tomto článku si popíšeme možnosti spolupráce TEXu a volně šiřitelného systému počítačové algebry Sage. Zaměříme se zejména na převod LATEXového dokumentu do formátu programu Sage a opačný převod z programu Sage do PDF prostřednictvím PDFLATEXu. Oba způsoby konverze jsou velmi mladé (závěr roku 2009), využívají však ve velké míře podrobně odzkoušené a odladěné programy, jako například program TEX4ht. Dále popíšeme možnosti a výhody volání programu Sage uvnitř LATEXového souboru protřednictvím balíčku SageTEX.

Klíčová slova: LATEX, Sage, Python, HTML, TEX4ht, matematika.

Představení programu Sage

Než přistoupíme k popisu způsobů spolupráce mezi programy TEX a Sage, program Sage si stručně představíme. Tento program je na své domovské stránce umístěné na `http://www.sagemath.org/` uveden následujícím popisem:

> *Sage is a free open-source mathematics software system licensed under the GPL. It combines the power of many existing open-source packages into a common Python-based interface.*
> *Mission: Creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab.*

Sage je velmi mladý systém počítačové algebry – je vyvíjen teprve od roku 2004. Přesto je již poměrně vyspělý a to zejména díky tomu, že v těch oblastech matematiky, které dosud nebyly do programu začleněny nativně, program pouze předá potřebné údaje jinému (třeba i úzce specializovanému) programu a zpracuje jeho výstup. Například řešení diferenciálních rovnic a řešení nerovnic je realizováno voláním programu Maxima[1]. I výpočet integrálů je realizován programem Maxima. Pokud však tento výpočet selže, je možno pouhým doplněním volitelného parametru příkazu `integrate` použít integraci jiným programem, včetně

*Zkoumání možností programu Sage při výuce, jeho spolupráce s programem LATEX a spoluúčast na vývoji programu `sws2tex` jsou podporovány grantem 131/2010 FRVŠ *Počítačová podpora výuky matematiky pomocí volně šiřitelného software.*

[1] `http://maxima.sourceforge.net/`

doi: 10.5300/2012-3/163

programu Mathematica (prostřednictvím volně dostupné webové služby Wolfram Mathematica Online Integrator[2] – výstup této služby je programem Sage automaticky zpracován a uživatel vidí pouze příslušnou primitivní funkci).

Zajímavé je i pracovní prostředí programu Sage. S programem Sage je možno pracovat buď voláním ze skriptu Pythonu, interaktivně v textové konzoli, nebo pomocí Sage Notebooku v okně internetového prohlížeče. Pro potřeby začátečníků je nejzajímavější právě poslední možnost – práce s programem Sage v prostředí internetového prohlížeče.

Pro přístup k zápisníku programu Sage Notebook (zkráceně Sage) stačí otevřít okno prohlížeče a připojit se na příslušný server. Tímto serverem může být `localhost` v případě instalace na Linuxu, server spuštěný ve virtuálním počítači v případě instalace ve Windows nebo kterýkoliv Sage server na Internetu, ke kterému má uživatel přístup. Protože jsou k dispozici i volně přístupné Sage servery [5], je možno s programem pracovat i bez instalace na lokální počítač.

Vzhledem k tomu, že práce probíhá v internetovém prohlížeči, nečiní první seznámení začátečníkům zpravidla nijak velké potíže. Absence klikacího grafického uživatelského rozhraní známého z programů Maple, Mathematica nebo wxMaxima je vyvážena automatickým doplňováním příkazů, které funguje podobně jako doplňování příkazů v Linuxové konzoli a dále propracovaným systémem nápovědy. V zápisníku je přítomen plugin s populárním editorem TinyMCE[3] pro vkládání textových komentářů.

V prostředí internetového prohlížeče je možno používat interaktivní prvky podobné známým mapletům, kdy uživatel mění vstupní veličiny nikoliv přímo v zápisníku programu Sage, ale ve formulářovém okně či pomocí posuvného táhla. Výstup se automaticky přizpůsobuje změnám na vstupu. Nejen tato vlastnost činí program Sage zajímavý jak pro vlastní výpočty a matematické experimenty, tak i pro výuku matematiky na středních a vysokých školách.

Vzhledem k rozšířenosti programu TEX pro psaní matematických textů se nabízí například otázka, jak je možno začlenit možnosti nabízené programem Sage do našich textů, ať se již jedná o učební materiály nebo výstupy výzkumu.

Formát souboru `sws`

Formát v jakém pracuje se soubory TEX je čtenářům dostatečně znám. Abychom pochopili možnosti spolupráce programů TEX a Sage, popíšeme si stručně, jak vypadá formát `sws` souborů, používaných programem Sage. Při práci v prostředí Sage Notebooku se veškerá data ukládají na serveru. Pomocí volby v horním menu lze tato data uložit na lokální počítač ve formě souboru `sws` a poté opět

[2] `http://integrals.wolfram.com/index.jsp`
[3] `http://tinymce.moxiecode.com/`

nahrát na libovolný server. Soubor `sws` je ve své podstatě komprimovaný archiv, po jehož rozbalení získáme zejména

- soubory `worksheet.html` a `worksheet.txt`, obsahující vstupy a výstupy programu Sage a případné komentáře vložené mezi tyto vstupy uživatelem,
- adresářovou strukturu, která obsahuje obrázky vytvořené programem Sage a případné další objekty.

Obsah souboru `worksheet.html` může vypadat například takto:

```
<p>V&yacute;počet <strong>derivace</strong></p>

{{{id=1|
diff(1/x,x)
///
-1/x^2
}}}

{{{id=3|
(_).show()
plot(x^2,(x,-1,1))
///
<html><div class="math">-\frac{1}{x^{2}}</div></html>
<html><img src='cell://sage0.png'></html>
}}}
```

Porozumět tomuto textu je snadné, zejména porovnáme-li jej se snímkem obrazovky na Obrázku 1 na straně 4. Z uvedeného příkladu je zřejmé, že zápisník programu Sage obsahuje

- vlastní značky pro oddělení vstupních a výstupních políček,
- HTML značky (zpravidla pouze omezenou množinu značek, vložených buď programem Sage, nebo editorem TinyMCE).

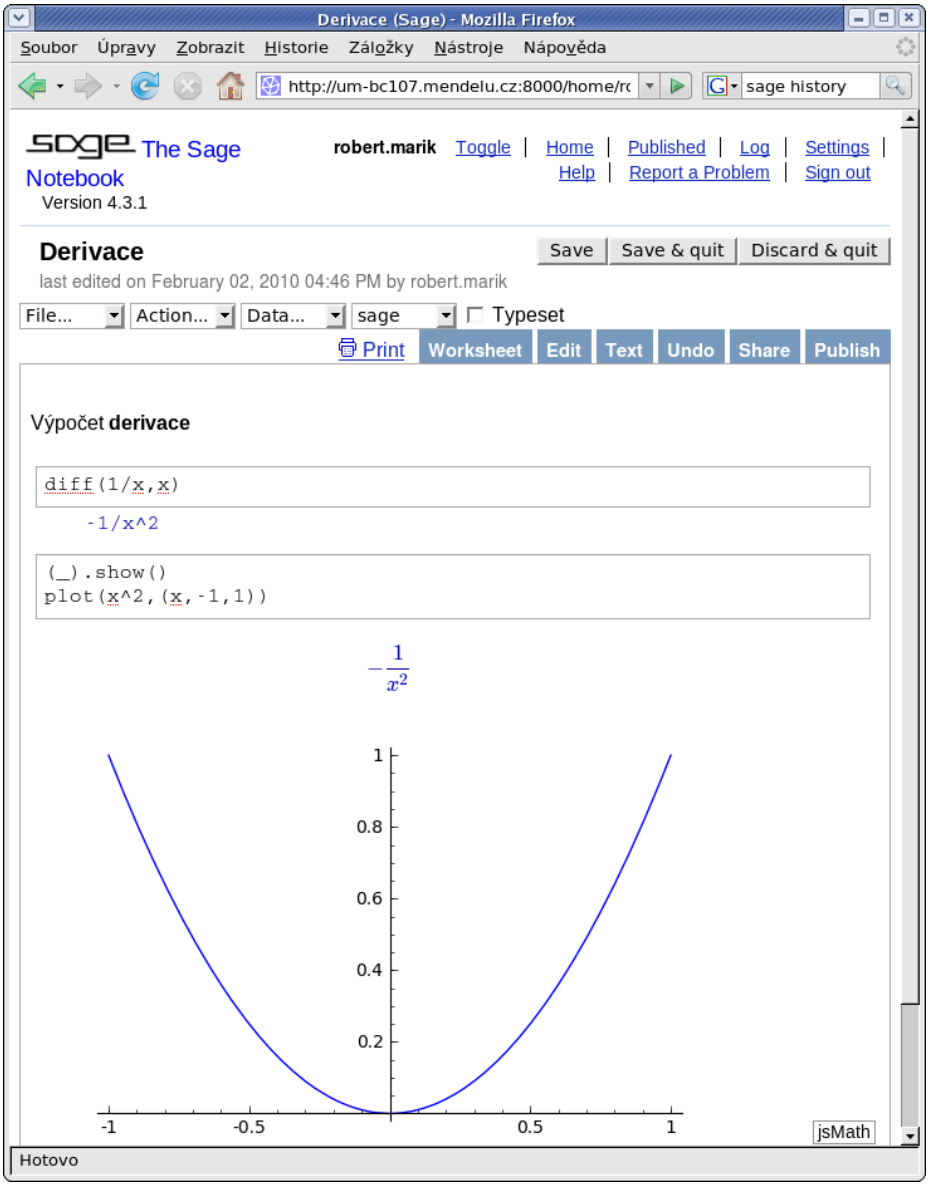
Při převodu mezi programy Sage a TEX se tedy jedná vlastně o konverzi mezi TEXem a HTML.

Z LATEXu do Sage

Konverze z LATEXu do HTML (a Sage) je založena podobně jako [4] na programu TEX4ht. Výsledný soubor je poté zpracován postprocesorem `tex2sws`[4], který na uživatelem definovaných místech vloží značky pro vstupní a výstupní buňky programu Sage, vytvoří potřebnou adresářovou strukturu a vše zabalí do formy `sws` souboru. Tento soubor je pak možno otevřít v programu Sage.

Takto vytvořený Sage zápisník bohužel obsahuje množství HTML tagů vložených programem TEX4ht. Například rovnice v display modu jsou zapsané jako tabulky

[4] `http://wiki.sagemath.org/devel/LatexToWorksheet`



Obrázek 1: Práce se zápisníkem programu Sage.

a editace a další práce s tímto textem není příliš pohodlná. Tato vlastnost je však jednak odstranitelná vhodnou konfigurací programu TEX4ht nebo vhodným rozšířením postprocesoru a není příliš omezující. Je totiž přirozené předpokládat, že čtenář takto vytvořeného materiálu či učebního textu bude převážně experimentovat s příkazy programu Sage a do textu nejvýše doplňovat své vlastní komentáře. Případné úpravy původního textu bude dělat autor v primárním LATEXovém zdrojovém souboru. Příkazy programu Sage jsou v LATEXovém zdrojovém souboru uloženy v prostředí `sageverbatim`[5]. Vložíme-li do LATEXového zdrojového souboru například text

```
Graf funkce nakreslíme příkazem \texttt{plot}:
\begin{sageverbatim}
  plot(x^2,(x,-1,1))
\end{sageverbatim}
Pro nastavení os a dalších parametrů obrázku
použijte příkaz \texttt{show}.
```

a zpracujeme-li jej následně programy TEX4ht a `tex2sws`, obdržíme soubor `sws`, který po načtení do programu Sage obsahuje náš text. Tento text je v místě prostředí `sageverbatim` přerušen a jsou sem vloženy vstupní a výstupní políčko programu Sage. Uživatel může pracovat klasicky jako v každém jiném zápisníku programu Sage, tj. může zejména

- editovat vstupní část políčka a spustit výpočet s jinými parametry (např. nakreslit graf jiné funkce),
- dvojklikem na textovou část vyvolat editor TinyMCE a upravit nebo doplnit doprovodný text dle svých vlastních potřeb.

V případě, že program TEX4ht rozdělí text do více souborů, obdržíme na výstupu programu `tex2sws` odpovídající počet vzájemně provázaných zápisníků.

Ze Sage do LATEXu

Další možnost spolupráce programů Sage a LATEX ocení především uživatelé, pro které je primárním dokumentem zápisník programu Sage a chtějí tento zápisník převést do LATEXu a následně do PDF. Tento převod je možno realizovat programem `sws2tex`[6]. Ve srovnání s klasickým tiskem do PDF souboru se tento způsob tvorby PDF vyznačuje

- vyšší kvalitou sazby, zejména u matematických výrazů[7],
- automatickým barevným zvýrazněním programového kódu pro lepší čitelnost,

[5] Viz například ukázka umístěna na `http://bitbucket.org/rbeezer/tex2sws/src/tip/example/example.tex`.

[6] `http://bitbucket.org/whuss/sws2tex/`

[7] Viz např. ukázky na `http://user.mendelu.cz/marik/sage/`.

- přibalením `sws` souboru k výslednému PDF, pro možnost snadno načíst text opět do programu Sage.

Protože `sws` soubor obsahuje HTML kód, není možné očekávat, že takto může být zpracován libovolný vstupní soubor. Nicméně na výstupu současné verze programu `sws2tex` obdržíme správné formátování textu (barva, typ písma, zarovnání odstavců, výčtová prostředí, jednotlivé úrovně nadpisů), obrázky v souborech `pdf`, `png` a `jpg`, kotvy a odkazy a také jednoduché tabulky.

Vstupní pole jsou formátovány přímo prostředím `verbatim`. Pro barevné zvýraznění programového kódu je použita knihovna `pygments`[8], v případě potřeby však je možno upravit program tak, aby spolupracoval s jiným zvýrazňovačem kódu, například s programem Highlight popsaném v jednom z nedávných Zpravodajů [3]. Výstupní pole a případná textová políčka jsou čtena funkcemi modulu `HTMLParser`[9] a použité tagy jsou nahrazovány odpovídajícím LaTeXovým kódem. Program `sws2tex` podporuje i konfigurační soubory, kde je možno definovat vlastní hlavičku vygenerovaného LaTeXového souboru, nastavit jazyk dokumentu, autora a další parametry.

LaTeX uvnitř Sage a Sage uvnitř LaTeXu

Systém Sage umožňuje přímé použití řady dalších programů. Uvedeme-li na prvním řádku vstupního pole příkaz `%octave`, je vstup zpracován programem Octave. Použijeme-li `%latex` následovaný kódem v LaTeXu, je tento vstupní kód předán programu LaTeX a následně programu `dvipng`. Sage zobrazí výsledek ve formátu `png`. Pokud řádku se specifikací programu určeného pro zpracování vstupu předřadíme ještě řádek `%hide`, zůstane vstup skrytý a vidíme jenom zpracovaný výstup.

Je možno pracovat i naopak a zapisovat příkazy programu Sage přímo do LaTeXového zdrojového souboru. Pro co největší pohodlí při tomto způsobu práce slouží balíček SageTeX[10]. Při kompilaci takového souboru TeXem je vytvořen pomocný soubor příkazů pro Sage. Tento soubor následně zpracujeme programem Sage a při další kompilaci TeXem již jsou výsledky těchto výpočtů vloženy na své místo. Práce je tedy podobná začleňování obrázků pomocí `mfpic`. Navíc SageTeX umí pracovat nejen s instalací Sage na lokálním počítači, ale je možno výpočty provádět i na některém z veřejných serverů [5]. Je dále vhodné poznamenat, že při použití SageTeXu má uživatel k dispozici interpreter programovacího jazyka Python. Tento jazyk je možno použít například k výrobě tabulek, což je demonstrováno v dokumentaci šířené se SageTeXem na příkladu Pascalova trojúhelníku. Tím se otevírá možnost poněkud odlišného přístupu k problému

[8] `http://pygments.org/`

[9] `http://docs.python.org/library/htmlparser.html`

[10] `http://www.ctan.org/tex-archive/help/Catalogue/entries/sagetex.html`

tvorby tabulek [2]. Je například také možno v LaTeXovém dokumentu použít Pythonovský kód uvedený v [1] k tvorbě Hornerova schématu. Autorem zmíněného kódu je pan Michal Kaukič. Tímto způsobem lze do dokumentu vkládat i obrázky, pro jejichž tvorbu Sage používá program `matplotlib`[11].

Praktické ukázky

Není bez zajímavosti ukázat použití představených nástrojů v praxi.

Konverzi programem `sws2tex` si ukážeme na příkladu zápisníku programu Sage s ukázkami řešení rovnic a nerovnic. Na Obrázcích 2 až 4 na str. 9 až 11 vidíme nejprve část zápisníku vytvořeného v programu Sage a poté výsledek po zpracování programem `sws2tex`.

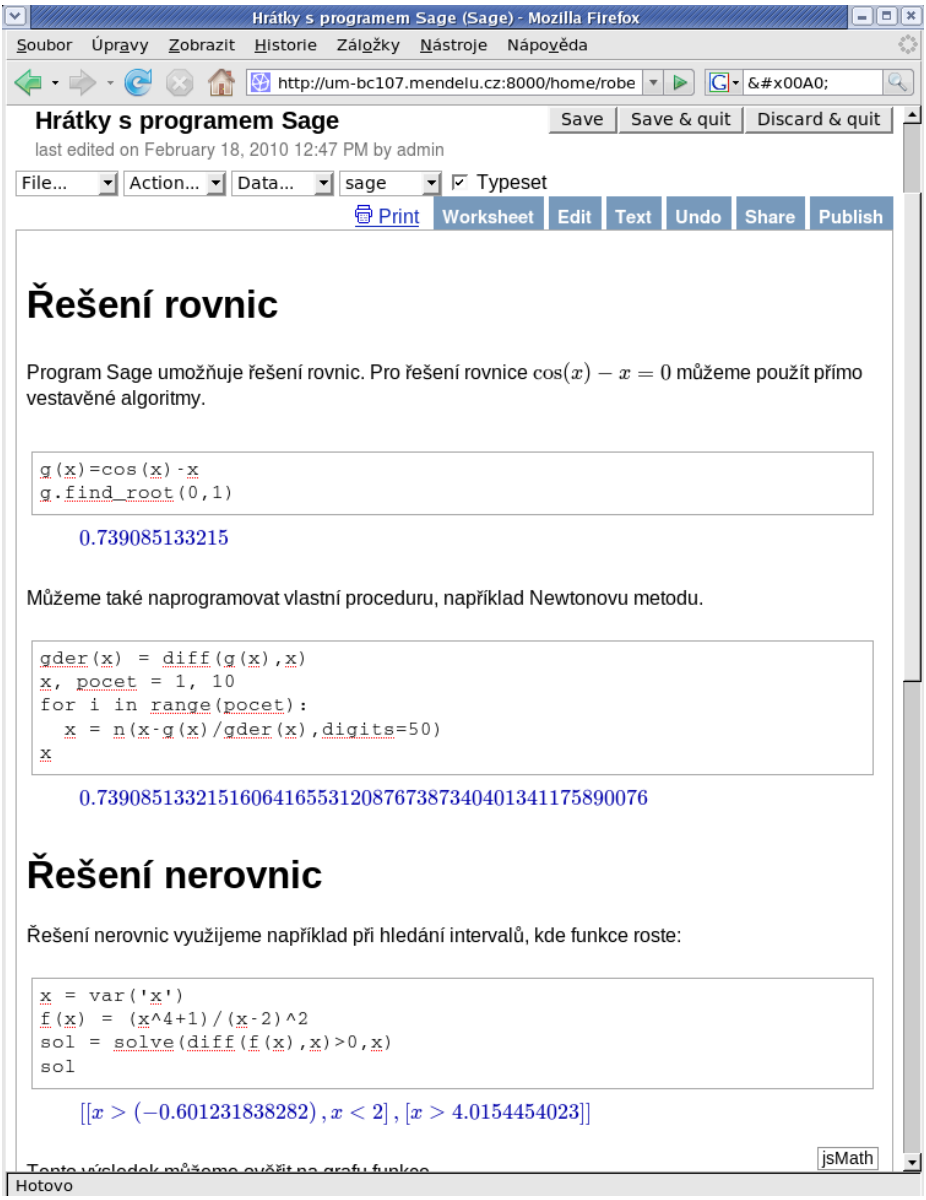
Pozn. Špendlík s připojeným `sws` souborem na těchto obrázcích není vidět, protože připojené soubory jsou příkazem `\includegraphics` ignorovány.

Soubor pro SageTeX s podobnou problematikou může vypadat například následovně. Všimněte si, že se jedná o klasický LaTeXový text obohacený o příkazy `\sage` (příkaz programu Sage), `\sageplot` (obrázek vytvořený v programu Sage) a prostředí `sagesilent` (příkazy programu Sage, které se však netisknou).

```
\documentclass{article}
\usepackage{sagetex}
\usepackage[margin=1in]{geometry}
\usepackage[T1]{fontenc}
\usepackage[latin2]{inputenc}
\usepackage[czech]{babel}
\begin{document}
\title{Hrátky s programem Sage}
\maketitle

\section{Řešení rovnic}
\begin{sagesilent}
g(x) = x-cos(x)
\end{sagesilent}

Program Sage umožňuje řešení rovnic. Pro řešení rovnice
$\sage{g(x) == 0}$ můžeme použít přímo vestavěné
algoritmy a obdržíme $x\approx\sage{g.find_root(0,1)}$.

Můžeme také použít vlastní proceduru, například Newtonovu metodu.
```

[11] `http://matplotlib.sourceforge.net/`

```
\begin{sagesilent}
gder(x) = diff(g(x),x)
x, pocet = 1, 10
for i in range(pocet):
  x = n(x-g(x)/gder(x),digits=50)
\end{sagesilent}
Výsledkem takového výpočtu po $\sage{pocet}$ iteracích
je hodnota $x\approx\sage{x}$.

\section{Řešení nerovnic}
Řešení nerovnic využijeme například při hledání intervalů,
kde funkce roste:
\begin{sagesilent}
x = var('x')
f(x) = (x^4+1)/(x-2)^2
sol = solve(diff(f(x),x)>0,x)
\end{sagesilent}
Funkce $f(x)=\sage{f(x)}$ roste na následujících
intervalech $\sage{sol}$.

Pro kontrolu si můžeme nakreslit graf funkce
$y=\sage{f(x)}$:

\begin{sagesilent}
P = plot(f(x),(x,-5,8),detect_poles=True)
\end{sagesilent}

\begin{center}
  \sageplot{P,ymax=100}
\end{center}

Hraniční body oddělující intervaly monotonie jsme získali
ve tvaru desetinných čísel. Nulové body derivace ale můžeme
vypočítat i přesně. Položíme-li v oboru reálných čísel
derivaci rovnu nule,
\begin{sagesilent}
sol_c = solve(diff(f(x),x)==0,x)
sol_r=[i.rhs() for i in sol_c if i.rhs().imag() == 0]
\end{sagesilent}
obdržíme body $\sage{sol_r}$. Numerickou aproximací obdržíme
následující body: $\sage{[i.n() for i in sol_r]}$.
\end{document}
```



Obrázek 2: Zápisník programu Sage.

Hrátky s programem Sage

Robert Mařík

18. února 2010

1 Řešení rovnic

Program Sage umožňuje řešení rovnic. Pro řešení rovnice $\cos(x) - x = 0$ můžeme použít přímo vestavěné algoritmy.

Sage code

```
g(x)=cos(x)-x
g.find_root(0,1)
```

0.739085133215

Můžeme také naprogramovat vlastní proceduru, například Newtonovu metodu.

Sage code

```
gder(x) = diff(g(x),x)
x, pocet = 1, 10
for i in range(pocet):
  x = n(x-g(x)/gder(x),digits=50)
x
```

0.73908513321516064165531208767387340401341175890076

2 Řešení nerovnic

Řešení nerovnic využijeme například při hledání intervalů, kde funkce roste:

Sage code

```
x = var('x')
f(x) = (x^4+1)/(x-2)^2
sol = solve(diff(f(x),x)>0,x)
sol
```

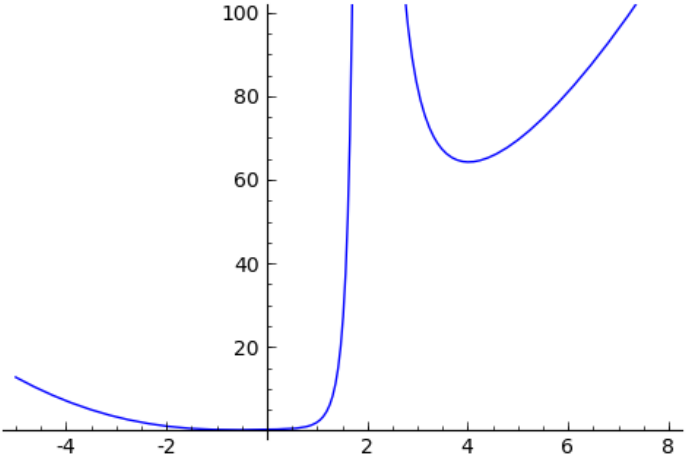
$[[x > (-0.601231838282)\,, x < 2]\,, [x > 4.0154454023]]$

Tento výsledek můžeme ověřit na grafu funkce

Sage code

```
plot(f(x),(x,-5,8),detect_poles=True).show(ymax=100)
```

Obrázek 3: Výstup programu **sws2tex**, strana 1.



Pro kontrolu můžeme ještě vypočítat nulové body derivace přesně. Položíme-li v oboru reálných čísel derivaci rovnu nule, obdržíme dva stacionární body:

Sage code

```
sol_c = solve(diff(f(x),x)==0,x)
sol_r=[i.rhs() for i in sol_c if i.rhs().imag() == 0]
sol_r
```

$$\left[-\frac{1}{2}\sqrt{8\sqrt{2}+10}+\frac{1}{2}\sqrt{2}+1, \frac{1}{2}\sqrt{8\sqrt{2}+10}+\frac{1}{2}\sqrt{2}+1\right]$$

Sage code

```
n(sol_r[0]),n(sol_r[1])
```

$$(-0.601231825852331, 4.01544538822543)$$

Obrázek 4: Výstup programu `sws2tex`, strana 2.

Dokument na Obrázku 5 na straně 12 obdržíme po zpracování (například) touto sekvencí příkazů:

```
pdflatex file.tex
sage file.sage
pdflatex file.tex
```

Závěr

V článku byly popsány možnosti spolupráce typografického systému TEX se systémem počítačové algebry Sage, zejména způsoby konverze dat a volání jednoho programu uvnitř druhého.

Hrátky s programem Sage

18. února 2010

1 Řešení rovnic

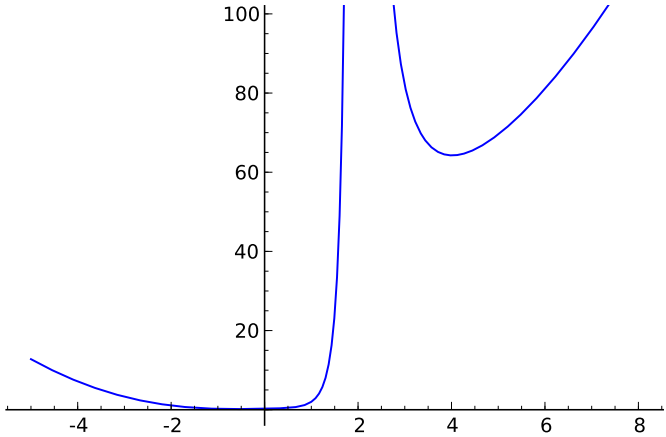
Program Sage umožňuje řešení rovnic. Pro řešení rovnice $x - \cos(x) = 0$ můžeme použít přímo vestavěné algoritmy a obdržíme $x \approx 0.739085133215$.

Můžeme také použít vlastní proceduru, například Newtonovu metodu. Výsledkem takového výpočtu po 10 iteracích je hodnota $x \approx 0.739085133215160641655312087673873404013411758900 76$.

2 Řešení nerovnic

Řešení nerovnic využijeme například při hledání intervalů, kde funkce roste: Funkce $f(x) = \frac{\left(x^4+1\right)}{(x-2)^2}$ roste na následujících intervalech $[[x > (-0.601231838282), x < 2], [x > 4.0154454023]]$.

Pro kontrolu si můžeme nakreslit graf funkce $y = \frac{\left(x^4+1\right)}{(x-2)^2}$:



Hraniční body oddělující intervaly monotonie jsme získali ve tvaru desetinných čísel. Nulové body derivace ale můžeme vypočítat i přesně. Položíme-li v oboru reálných čísel derivaci rovnu nule, obdržíme body $\left[-\frac{1}{2}\sqrt{8\sqrt{2}+10}+\frac{1}{2}\sqrt{2}+1, \frac{1}{2}\sqrt{8\sqrt{2}+10}+\frac{1}{2}\sqrt{2}+1\right]$. Numerickou aproximací obdržíme následující body: $[-0.601231825852331, 4.01544538822543]$.

Obrázek 5: Výstup programu SageTEX.

Seznam literatury

[1] Blaško, Rudolf. LaTeX a neobvyklé výpočty v pevnej rádovej čiarke. [LaTeX and Unusual Fixed Point Computations.] *7th International Conference Aplimat* (2008) [7. mezinárodní konference Aplimat], Part IV – Open Source Software in Research and Education, pp. 9–18. ISBN 978-80-89313-04-4. Též online: `http://www.sospreskoly.org/latex-a-neobvykle-vypocty-v-pevnej-radovej-ciarke`

[2] Hlávka, Zdeněk. Velkovýroba tabulek pomocí AWK. [Large-scale Production of Tables in AWK.] *Zpravodaj Československého sdružení uživatelů TEXu* [The Bulletin of the Czechoslovak TeX Users Group], Vol. 18 (1–2), pp. 93–95, 2008. ISSN 1211–6661.

[3] Simon, André. The Highlight programme: code & syntax highlighting. [Program Highlight a jeho využití.] *Zpravodaj Československého sdružení uživatelů TEXu* [The Bulletin of the Czechoslovak TeX Users Group], Vol. 19 (4), pp. 222–239, 2009. ISSN 1211–6661.

[4] Sojka, Petr; Růžička, Michal. Publikování z jednoho zdroje v odlišných formátech pro různá vstupní zařízení. [Parallel Electronic Publications.] *Zpravodaj Československého sdružení uživatelů TEXu* [The Bulletin of the Czechoslovak TeX Users Group], Vol. 18 (3), pp. 116–129, 2008. ISSN 1211–6661.

[5] Stein, William et al. *The Sage notebook.* [online cit. 16. 2. 2010] URL: `http://www.sagenb.org/`, `http://sagenb.kaist.ac.kr/`

Summary: Cooperation between TeX and computer algebra system Sage

In this paper we describe recent progress in cooperation between TeX and open-source computer algebra system Sage. The paper is focused on possibilities to call one program inside the other one and on data conversion.

Keywords: LaTeX, Sage, Python, HTML, TeX4ht, mathematics.

Robert Mařík, marik@mendelu.cz
Ústav matematiky, Lesnická a dřevařská fakulta
Mendelova univerzita v Brně, Zemědělská 1
CZ-621 00 Brno, Czech Republic

TEX v jednoduchém unixovém prostředí

Petr Olšák

Abstrakt:

Při ladění TEXového dokumentu potřebujeme mnohokrát opakovaně pouštět TEX, podívat se, jak dopadl výsledek v prohlížeči DVI nebo PDF souboru, mrknout na výpis TEXu na terminálu, podívat se případně do logu a celou činnost opakovat. V tomto článku je ukázáno, jak tuto práci dělá autor článku. Proces „editor-TEX-kuk“ je zde podporován jednoduchými unixovými nástroji: bashovým skriptem `texloop`, který si autor pro tyto účely vytvořil, dále terminálem Xterm a jednoduchým editorem, který umí navázat na klávesovou zkratku spuštění příkazu v systému. Čtenář se zde může inspirovat a přizpůsobit tyto nástroje svým vlastním potřebám. V článku je popsána funkce skriptu `texloop`, dále je neformálně rozveden dlouholetý vývoj autorova vztahu k textovým editorům a konečně je zde uvedena konfigurace terminálu Xterm, aby vyhovoval českému prostředí jak v kódování ISO-8859-2, tak v kódování UTF-8. Pro kódování UTF-8 si v závěru článku vygenerujeme TEXový formát `csplain`.

Klíčová slova: skript `texloop`, Unix, editor Joe

Skript `texloop`

Bashový skript `texloop` se chová jako démon ve smyslu, že čeká na signály. Ovšem protokol o své činnosti vypisuje na terminál. Můžeme ho tedy pustit v nějakém terminálu a v jiném terminálu pustit textový editor. Pokud v editoru uložíme zdrojový soubor a vyšleme signál (to můžeme udělat stiskem jediné klávesy, stačí si na to připravit klávesovou zkratku v editoru), démon `texloop` se probudí, spustí požadovanou TEXovou úlohu, a přitom na „svůj“ terminál vypisuje hlášení TEXu. Pokud úloha dopadla bez chyby, pošle démon signál prohlížeči DVI nebo PDF a ten automaticky obnoví zobrazení. Takže na jedinou klapku v editoru TEXujeme a hned vidíme výsledné změny v prohlížeči.

Démon `texloop` spustíme příkazem

```
texloop tex-command main-file
```

kde „`main-file`“ je název TEXovaného souboru bez přípony, například tedy

```
texloop csplain mujtext
```

 doi: 10.5300/2012-3/176

V první fázi činnosti je `texloop` v interaktivním módu. Spustí TEX podle zadaného příkazu se zadaným vstupním souborem. Pokud TEX narazí na chybu, zastaví se a čeká na terminálu na reakci. Komunikaci můžeme samozřejmě ukončit pomocí všem TEXistům známé klávesy `X`. Ovšem to uděláme s tím rizikem, že nemusí vzniknout soubor pro prohlížení. Jak se v takovém případě `texloop` zachová, popíšu za chvíli.

Souborem pro prohlížení je soubor `main-file.pdf` v případě, že `tex-command` začíná na písmena `pdf`. Například tedy `pdftex`, `pdfcsplain`, `pdflatex`. Jinak je souborem pro prohlížení `main-file.dvi`.

Pokud existuje po prvním interaktivním běhu TEXu soubor pro prohlížení, `texloop` spustí odpovídající prohlížeč tohoto souboru (`xpdf` nebo `xdvi`) a přejde do „daemon“ módu, kdy čeká na signály. Když soubor pro prohlížení neexistuje, žádný prohlížeč se zatím nespustí a `texloop` i v tomto případě přejde do „daemon“ módu. Pokud `texloop` po některém z příštích běhů TEXu zjistí, že soubor pro prohlížení dodatečně vznikl, neváhá v daném okamžiku spustit odpovídající prohlížeč.

V „daemon“ módu `texloop` čeká na signály. Rozlišuje pouze dva: `SIGUSR1` pro spuštění TEXu a `SIGTERM` pro ukončení činnosti. Obdrží-li signál `SIGUSR1`, pak zkontroluje časy poslední modifikace souborů `main-file.tex` a `main-file.log`. TEXová úloha se spustí právě tehdy, když je zdroják novější než log. Výjimku z tohoto pravidla uvedu za chvíli.

TEXová úloha v „daemon“ módu běží v neinteraktivním režimu. Při výskytu první chyby TEX ukončí svoji činnost. Možnost pokračování činnosti TEXu a výpisu všech chyb jsem zavrhnul, protože dnes jsou počítače natolik rychlé, že nikoho nezdržuje po opravení první chyby spustit TEX znovu. Osobně tento režim práce preferuji.

Jestliže běh TEXu skončil úspěšně, `texloop` požádá prostřednictvím signálu prohlížeč TEXového výstupu, aby znovunačetl PDF resp. DVI a obnovil zobrazení. V každém případě po ukončení běhu TEXu přechází `texloop` znovu do stavu čekání na signály.

Proces `texloop` končí svou činnost buď po obdržení signálu `SIGTERM` (můžeme zmáčknout `Ctrl-C` v terminálu, kde `texloop` běží), nebo ukončením prohlížeče PDF nebo DVI prohlížeče, který byl procesem `texloop` spuštěn.

Někdy se může stát, že editujeme jiný soubor, než „hlavní“ soubor `main-file.tex`. Představme si, že je v `main-file.tex` příkaz `\input` a ten načítá nějakou kapitolu dokumentu. Textovým editorem jsme zrovna zalezlí do této kapitoly, měníme ji a chceme vidět výsledek zpracování celého dokumentu. Uložení kapitoly a vyslání signálu pro `texloop` nepomůže, protože `texloop` zjistí, že je log novější než zdroják a na TEXování se vydlábne. Můžeme ovšem potlačit vykonání testu, kterým `texloop` kontroluje stáří zdrojáku a logu. Stačí přidat k parametrům pro `texloop` písmeno `A` (jako „run TEX Always“). Například tedy

```
texloop csplain kniha A
```

Abychom mohli daemonu `texloop` vyslat signál, musíme vědět, jaký má PID. Program `texloop` v okamžiku přechodu z interaktivního do „daemon“ módu uloží svůj PID do souboru `~/.tlpid`, odkud si jej může editor přečíst. Když ovšem spustíme další proces `texloop` (vedle už běžícího), pak nový proces přepíše obsah tohoto souboru svou hodnotou. Editor si tedy musí „zavčas“ informaci převzít. Druhá možnost je, že editor vyšle signál všem procesům `texloop` daného uživatele a konkrétním číslem procesu se nezabývá. Pak stačí, když použije příkaz `killall -USR1 texloop`. Nevýhoda tohoto přístupu je, že možná některé procesy budou TEXovat zbytečně, ale není to obvyklé. Procesy bez příznaku `A` (Always) totiž budou TEXovat jen v případě, že je zdroják novější než log.

Skriptík `texloop` je docela jednoduchý. Má 80 řádků, najdete ho na [1] a můžete si ho přiohnout k obrazu svému.

Textové editory, aneb konečně končím s Emacsem

V této kapitole rozvedu neformálním způsobem své zkušenosti s textovými editory. Pro lidi, kteří očekávají spíše nezaujaté technické informace a zejména pro fanoušky Emacsu není tato kapitola určena. Pro uživatele MS Windows také není tato kapitola určena. Tito uživatelé jistě už poznali, že jim je k ničemu kompletně celý článek.

S interaktivními textovými editory jsem se poprvé setkal v operačním systému DOS a poté v Unixu. Jiné operační systémy umožňující interakci s uživatelem jsem neměl tu čest poznat.

V operačním systému DOS jsem po dlouhém vývoji zakotvil u Qeditu, který jsem si přizpůsobil obrazu svému. Jeho výchozí chování bylo zřejmě inspirováno editorem WordStar. Qedit měl přehledný konfigurační soubor s možností jednoduché tvorby maker a klávesových zkratek. Ovšem to bylo zhruba před dvaceti lety. Při přechodu na Unix jsem se naučil aspoň základy editoru `vi`. Tento editor měl totiž tu vlastnost, že se dal najít na skoro každém Unixu, i kdyby tam už nebylo vůbec nic jiného. Setkal jsem se výjimečně i s Unixy natolik ořezanými, že tam nebyl ani `vi`. Editovat konfigurační soubory systému v řádkovém editoru `ed` pak byla skutečná lahůdka. Tehdy nás ale nic nemohlo odradit od našeho nadšení z Unixu.

Až do roku 2011 jsem používal dva editory: `vi` a Emacs. První jmenovaný jsem používal jako „odlehčený editor“ pro editaci systémových souborů. „Odlehčený editor“ musí být snadno dostupný v každém Unixu a musí umět spolupracovat s běžně používanými terminály. Nesmí se opírat o přítomnost Xserveru, protože ten často není při konfiguraci systému přítomen. Umím jen úplné základy editoru `vi` a podobně jako plno jiných lidí mě psychicky nesmírně vyčerpává přepínat mezi dvěma módy tohoto editoru. Ale co, v době počátku Unixů nic jiného nebylo.

Druhý jmenovaný editor (Emacs) je kapitola sama pro sebe. Před dvaceti lety měla jeho uživatelská dokumentace 700 stránek, dnes jsou k Emacsu dokumenty dva, dohromady mají 1600 stránek. Tím se mi nikdy nepodařilo prokousat. Komu se to podařilo, nechť se přihlásí a pak hodí kamenem. Domnívám se, že editor má sloužit a ne být středem pozornosti a nutit uživatele mu věnovat neúměrnou pozornost. Emacs má velmi neobvyklý způsob ovládání, který s mým oblíbeným Qeditem z OS DOS má pramálo společného. Také používá naprosto neobvyklou terminologii, takže najít něco konkrétního v dokumentaci metodou `grep` je téměř nemožné. Nevíte totiž, co máte hledat. Navíc je to komplikováno tím, že Emacs má dokumentaci uloženu ve svém svérázném formátu, ke kterému nabízí přístup svým svérázným způsobem (programem `info`). Uvedu namátkou jen tři terminologické svéráznosti. Klávesovou zkratku `Ctrl-A` Emacs značí „`C-a`“. Klávesu `Alt` nazývá „Meta“ a značí „`M`“. Symbol „`M-A B`“ v dokumentaci znamená „zmáčkni `Alt` a `Shift`, zmáčkni `A`, pusť `Alt` a stále drž `Shift`, zmáčkni `B`“. Kdo nepustí klávesu `Alt` ve správném okamžiku, má smůlu, Emacs vykoná něco jiného. Nikomu není příjemné, když se mu Emacs takto směje do ksichtu.

Protože Emacs byl v té době jediný editor, který byl použitelný na rozsáhlejší věci (včetně TEXování rovnou z editoru), investoval jsem do něj asi měsíc svého života a pokusil se vytvořit alespoň pár vlastních klávesových zkratek na TEXování a pro aspoň trochu normální ovládání. I poté jsem některé složitější úkoly (např. posuny sloupcových bloků) v Emacsu udělat neuměl. Další čas strávený nad dokumentací, než bych dokázal potřebnou informaci vyhledat, se mi hrubě nevyplatil. Pokorně jsem tedy pustil ve vedlejším okénku emulátor OS DOS, v něm nahodil Qedit a úkol vyřešil tam.

Asi před deseti lety přišla další nepříjemnost v podobě přechodu z verze Emacsu 19 na verzi vyšší. Mnou pracně vytvořená makra v nové verzi samozřejmě nefungovala. Dokumentace se stala ještě obludnější a nesrozumitelnější. Nějakou dobu jsem kromě implicitní verze Emacsu v unixové distribuci udržoval vedle i starou verzi 19, na které jsem provozoval veškerou svou práci. Bohužel, jak roky běžely, byla verze 19 stále obtížněji kompilovatelná pro nové distribuce. Zpočátku jsem se dokázal povrtat v Céčkových zdrojích a kompilaci nějak rozdýchat, ale s příchodem dalších verzí kompilátoru a zejména novějších knihoven to šlo stále hůř. Takže jsem se rozhodl znovu investovat pár týdnů svého času, ponořit se do zrůdné dokumentace a vytvořit pokud možno tytéž klávesové zkratky, jako jsem používal pro Emacs 19. Od jednoho emacsového guru jsem se dozvěděl další nepříjemnost: nová verze už nebude umět jedním klapnutím opravit špatně napsaný kus řádku, který člověk hledící do klávesnice (místo na monitor) napsal omylem v nespr8vn2 p5epnut0 kl8vesnici. Cituji názor tohoto emacsového guru: naučte se psát všemi deseti a dívat se při psaní na monitor.

Všechny unixové editory, se kterými jsem se setkal, trpí dalším neduhem: předpokládají, že člověk sedí u vyorané klávesnice z doby kamenné, která nemá funkční klávesy na pohyb kurzoru. Takže nejstručnější klávesové zkratky řeší

posun kurzoru. Například `Ctrl-F` je posun o jedno políčko doprava. Ha, jak skvělé, ale jak k ničemu! Začátečnický tutoriál k Emacsu je rovněž plný takových totálně zbytečných informací. I tohle komplikuje vztahy mezi editorem a jeho potenciálními uživateli.

Měl jsem možnost mluvit s jedním příznivcem Emacsu. Zeptal jsem se ho, jak se dá přepnout wordwrap mód z implicitního emacsího chování, kdy je potřeba psát dlouhé slovo poslepu třeba za roh a teprve mezera za slovem toto slovo ulomí na další řádek. Očekával bych normální chování, kdy se slovo ulomí v okamžiku, kdy kurzor dosáhne nastavené hranice. Dále jsem se chtěl nechat poučit, jak efektivně v dokumentaci zjistit některé věci. Třeba jak vypnout case-insensitive mód při povelu `isearch`. Nebo jak zkrotit syntaktické zvýrazňování. Pro TEXové zdrojáky přišel Emacs s novinkou, že texty za podtržítkem se zobrazují zmenšeně (jako indexy). To působí velmi rušivě. Jak to opravit a nepřijít o zvýrazňování? Ani na jednu otázku jsem nedostal uspokojivou odpověď.

Emacs je z mého pohledu obluda, která se vymyká **základnímu zákonu Unixu**, který praví: *Unixové prostředí sestává z malých elementárních programů, které dělají jen jeden dílčí úkol, kvůli kterému byly navrženy, a ten dělají dobře. Tyto programy vzájemně spolupracují.* Mám na mysli třeba `grep`, `awk`, `bash`, `bc`, `tar`, `sed`, `rsync`, TEX a další. Editor Emacs byl prvním významným narušitelem tohoto základního zákona. Pak přišli další, o OO se nebudu raději zmiňovat.

Z výše uvedeného vyplývá, že jsem při editování souborů dvacet let trpěl. Dokonce jsem pomýšlel na to, že si napíšu editor vlastní. Ale letos přišlo vysvobození. Začalo to tím, že jsem si instaloval nově systém do notebooku. Jako první potřebuji mít odlehčený editor na editování konfiguračních souborů systému. V Gentoo Linuxu je pro tento účel přednastaven editor `nano`, který ale člověka pravidelně vytáčí dvěma hloupými otázkami, než uloží pozměněný soubor a ukončí činnost. Přeci jenom `vi` editor se svým `Shift-ZZ` vykoná tuto práci bez ptaní a hned. Editor `vi` v čisté podobě už dávno neexistuje, používal jsem tedy už dlouho jeho `vim` implementaci. A v nové verzi mě `vim` velmi rozhněval. Implicitně má nastaveno na kurzorové klávesy vlevo/vpravo chození po slovech. V tomto režimu jsem vůbec nebyl schopen editovat. Řekl jsem si, že pokud do deseti minut v dokumentaci či pomocí Google nenajdu, jak se tato nová skvělost odstraňuje, letí `vim` z mého počítače pryč. A taky že letěl.

Čím ale `vi` nahradit? Před deseti lety jsem viděl v provozu editor Joe, takže vím, že to je další možný odlehčený editor. Během pár vteřin po jeho nainstalování jsem věděl, že bez hloupého ptaní uložím soubor a ukončím činnost editoru pomocí `Ctrl-KX`. Během pár minut jsem shledal, že editor za těch deset let výrazně pokročil kupředu. Během hodiny už začínám tušit, že toto je editor, který mi nahradí nejen odlehčený editor, ale také Emacs. Má všechny funkce, které pro své editování potřebuji. Vše jsem se dozvěděl v kratičké manové stránce, online nápovědě a v přehledně vyvedeném vzorovém konfiguračním souboru `/etc/joe/joerc`. Jako třešničku na dortu uvedu skutečnost, že Joe má 25krát

menší binárku než Emacs. A dělá přesně to, k čemu byl určen, a dělá to dobře. V souladu se **základním zákonem Unixu**. Spočítejte si, kolik knihoven jednotlivé binárky načítají dynamicky. Emacs jich potřebuje ke svému životu 44, zatímco editoru Joe stačí tři základní knihovny (`libc`, `libm` a `libutil`). Při kompilaci se dá nastavit ještě závislost na `ncurses`, ale není to nutné.

Níže uvádím seznam vlastností, které mě u editoru Joe potěšily. Mnohé z nich jsem nikdy nedokázal v Emacsu nastavit. Netvrdím, že by to v Emacsu nešlo, vždyť „Emacs umí všechno“. Spíš je to kvůli mé neschopnosti se orientovat v nepřehledné emacsí dokumentaci.

- Při rychlém hledání i při najdi-nahraď vnímá rozdíly mezi velkými a malými písmeny. Dá se to podle potřeby vypnout.
- Kurzor se nesnaží lepit na konce řádků, takže při posunování dolu/nahoru neskáče při své pozici vpravo jako čamrda.
- Při rolování dolů/nahoru obraz s sebou neškube.
- Wordwrap se chová normálně.
- Snadno přepnu do hexa módu.
- V UTF8 terminálu snadno editor přepnu do stavu, kdy edituje ISO-2 soubory a naopak v ISO-2 terminálu zase mohu editovat UTF8 kódované soubory.
- Neřeší, co řešit nemusí: fonty, přepínání klávesnice. To je starost terminálu.
- Snadno přeskočím během editování na místo v souboru, které jsem nedávno opustil a pak se zase snadno vrátím tam, kde jsem byl předtím.
- Editor si pamatuje polohu kurzoru před ukončením. Když vlezu do souboru po nějaké době znovu, startuje v místě, kde jsem soubor opustil. Taky si trvale pamatuje použité příkazy pro překlad a další věci. Historií těchto příkazů mohu procházet šipkami nahoru/dolů.
- Neptá se hloupě na každou pitomost.
- Únik ze všeho pomocí `Ctrl-C` je přirozený. Výchozí klávesové zkratky se podobají zkratkám v mém starém dobrém Qeditu (tj. jsou odvozeny z WordStaru).
- Má jednoduchou konfiguraci nových zkratek a maker. Během chvíle jsem si například nastavil propojení editoru s filtrem `vlna`, což byl úkol, na který jsem z hlediska jeho obtížnosti v případě Emacsu po celých dvacet let rezignoval a vlnkoval soubory „ručně“ ve vedlejším terminálu.
- Zobrazí `^M` na koncích řádků. Setkávám se totiž občas se soubory majícími pomršené konce řádku (z OS DOS nebo z MS Windows). Pokud se mě editor snaží odstínit od tohoto problému, je to špatně, neboť pak vznikají záhadné chyby například v shellových skriptech. Na druhé straně je editor schopen se přizpůsobit (když chci) a editovat v „dosovém“ módu konců řádků.
- Konečně zase umím snadno označit sloupcové bloky (i myší) a posunovat s jejich obsahem.
- Syntaktické zvýrazňování je příjemné a má jednoduché a přehledné konfigu-

rační soubory. Během chvíle jsem si vylepšil například syntaktické zvýrazňování TEXových zdrojáků k obrazu svému.
- Editor se naučí slova použitá v načteném souboru a umožní jejich rychlé doplňování. Stačí napsat začátek slova a zmáčknout příslušnou klávesu.

Nebudu zde uvádět všechny podstatné vlastnosti editoru. Stručný seznam jeho možností najdete na [2]. Myslím si, že editor nabízí vše, co běžný uživatel (jako jsem já) potřebuje pro normální práci.

Některé nevýhody editoru mohou plynout z faktu, že je zcela závislý na prostředí terminálu, ve kterém je spuštěn. Na vstupu tedy snímá ty informace, které mu tam pošle terminál. Například Xterm vysílá při zmáčknutí klávesy Backspace stejný kód jako při zmáčknutí klávesy Delete a to je stejný kód jako při `Ctrl-H`. Mezi těmito třemi klapkami pak editor nemůže rozlišit a reaguje na všechny jako Backspace. Chtěl bych vidět toho člověka, který vyvíjel termcap pro Xterm, když tam navrhnul Delete=Backspace. Asi před tímto rozhodnutím se mu srazila hlava s železničním pražcem. Problém jsem si opravil úpravou konfigurace Xtermu, ale od této chvíle nemám Xterm splňující oficiální parametry.

Klávesové zkratky `Alt-něco` se nedají použít, neboť je terminál převede na jednobytovou informaci typicky shodnou se zmáčknutím nějaké klapky s akcentovaným znakem. Editor se vůbec nedozví, že byla zmáčknuta klávesa `Alt`. Znamená to, že můžeme pracovat pouze s klávesovými zkratkami typu `Ctrl-něco`. Aspoň se to lépe pamatuje. V případě zkratek `Ctrl-něco` máme k dispozici jen 32 možností, při kterých terminál generuje kódy 0 až 31 takto: `Ctrl-@` je nula, `Ctrl-A` je 1, `Ctrl-B` je 2, atd., `Ctrl-^` je 30 a `Ctrl-_` je 31. Kdo zatím nevidí souvislost, podívá se za domácí úkol na ASCII tabulku. Přitom ještě platí, že `Escape`=`Ctrl-[`. Máme ovšem štěstí, že klávesy `F1`, `F2` atd. a některé další vysílají z terminálu k aplikaci poněkud delší sekvenci znaků (uvozenou kódem Escape), podle které aplikace rozpozná, co bylo zmáčknuto. Při zmáčknutí `Ctrl-F1` je vygenerována jiná escape sekvence než při `F1`. Takže tyto zkratky typu `Ctrl-Fněco` dokáže editor rozpoznat a dokonce nespadají do omezujících 32 možností zmíněných dříve.

Různé terminály vysílají bohužel při mačkání kláves F1 a spol. různé escape sekvence. Někdy se tyto sekvence dokonce překrývají v různém významu. Například RxVT a Xterm mají vzájemně prohozeny Home a End. Editor Joe bohužel neumí pracovat s konfigurací escape sekvencí závislou na použitém terminálu. Já používám výhradně Xterm a Linuxovou konzoli a pro tyto dva typy terminálů jsem si upravil svou konfiguraci editoru Joe.

Výše popsané omezení plyne z toho, že chceme použít odlehčený editor uvnitř terminálu, tj. třeba i na vzdáleném počítači s mizerným síťovým připojením. Zkuste si Emacs pustit uvězněný uvnitř terminálu a shledáte, že se za jistých okolností chová stejně. Žádnou klávesu `Meta` alias `Alt` nerozliší.

Další možnou nevýhodou editoru Joe je jeho až příliš jednoduchá konfigurace maker. Jsou k dispozici jen jednoduché podmínky, není možná žádná práce

s proměnnými, jen sled za sebou jdoucích elementárních pokynů pro editor. Abych tedy mohl v Joe příjemně TEXovat, vytvořil jsem si (mimo jiné) bashový skriptík `texloop`.

Někdo může za nevýhodu editoru Joe považovat také skutečnost, že se jeho autor asi před dvěma lety vytratil z Internetu. Když jsem mu nedávno posílal návrh na jednoduchou změnu ve formě záplaty, pošta se mi vrátila jako nedoručitelná kvůli přeplněné emailové schránce. Osobně by mi vůbec nevadilo, kdyby se vývoj editoru zastavil. Vychytávky, které si v konfiguraci editoru udělám, pak budou fungovat navždycky.

Zdrojové kódy editoru Joe jsou napsány ve srozumitelném jazyce C, takže jsem si sám dokázal opravit jednu drobnou chybku a doprogramovat reakci na prostřední tlačítko myši. Chtěl bych zdůraznit, že toto je výhoda *jednoduchého* a *otevřeného* software. Uživatel si jej může přiohnout k obrazu svému. Musí ale být splněny současně obě dvě vlastnosti. Například zdrojáky Otevřené kanceláře jsou sice otevřeny, ale vykostit z nich algoritmy na konverzi kancelářských formátů do PDF a udělat z nich dávkové programy pro příkazovou řádku se ještě nikomu nepodařilo.

Tím, že jsem opustil Emacs a `vim` a přešel konečně na normální editor, jsem si výrazně ulehčil život. Vymizela schizofrenie, zda v dané chvíli chci TEXovat nebo editovat konfigurační soubory vzdáleného serveru. Dělám to od této chvíle stejným editorem s jednotným způsobem ovládání. Autor editoru Joe Joseph H. Allen se přesně strefil do mých představ, jak má tento typ programů vypadat. Možná by měl mít tento článek jiný název: The Joy Of Joe.

Uživatelská konfigurace editoru

Uživatelská konfigurace editoru Joe je uložena (přirozeně) v souboru `~/.joerc`. Tam si můžeme definovat výchozí hodnoty přepínačů, jednoduchá makra a vazby na klávesové zkratky. Kromě toho je možné založit adresář `~/.joe` a do něj vložit další soubory konfigurace: konfiguraci závislou na různých typech souborů, konfiguraci syntaktického zvýrazňování pro různé jazyky apod. Při tvorbě uživatelské konfigurace je možné se inspirovat výchozí konfigurací `/etc/joe/joerc` nebo použít konfiguraci, kterou jsem si vytvořil pro svou potřebu a kterou jsem zveřejnil na [1]. V této kapitole stručně popíšu vlastnosti této uživatelské konfigurace.

Klávesové zkratky jsou trojího typu: „jednoklapkoidní“ (např. `Ctrl-C`), „dvouklapkoidní“ (např. `Ctrl-KX`) a zkratky uvozené klávesou Escape (např. `Esc L` – spuštění korekce překlepů pomocí programu `aspell`). Všechny jednoklapkoidní zkratky, které v originální konfiguraci řeší pohyb kurzoru, jsem považoval za neobsazené, protože pohyb kurzoru realizuji pomocí šipek a pomocí kláves Home, End, PgUp, PgDn. Některé z takto uvolněných `Ctrl-S` – rychlé vyhledávání vpřed, `Ctrl-R` – rychlé vyhledávání vzad,`Ctrl-Z` – undo, `Ctrl-U` – vyvrhnutí naposledy smazaného textu na místo kurzoru, `Ctrl-P` – skok na předchozí (vzdálenou)

pozici kurzoru, `Ctrl-J` – zrušení označeného bloku. Přesný přehled všech těchto změn je v krátkém dvojstránkovém shrnutí na [1].

Editor umožňuje (kromě maker v souboru `.joerc`) nahrávat další uživatelská makra `Ctrl-Kčíslo`. Někdy se ale hodí přehrát makro jednoklapkoidním hmatem, abychom je mohli přehrávat se stejnou rychlostí, na kterou máme nastaven opakovací kmitočet klávesnice. Proto jsem zkratku `Ctrl-K\` navěsil na přehrání makra 0.

Dvouklapkoidní zkratky typu `Ctrl-Kněco` jsem nechal v původním významu. Jsou odvozeny z WordStaru a já jsem se s nimi dávno seznámil při práci s Qeditem. Taktéž zkratky uvozené klávesou Escape jsem nechal v původním významu. Je tedy možné je dohledat v online nápovědě editoru, která je dostupná pomocí `Ctrl-KH`. Na další stránku nápovědy se přechází pomocí `Esc tečka`. To sice není příliš praktické, ale zase tak často v nápovědě listovat nepotřebujeme.

Nově jsem přidal dvouklapkoidní zkratky typu `Ctrl-Xněco`. Zkratka `Ctrl-XC` uloží vyznačený blok do souboru `~/.joetmp` a následně zkratka `Ctrl-XV` si obsah tohoto souboru vyzvedne. Tím je možné zprostředkovat přenos bloku mezi dvěma různými instancemi editoru Joe spuštěnými na stejném počítači.

Přenos bloku je samozřejmě možné provést i pomocí clipboardu X windows systému. Ovšem pozor. V konfiguraci mám zapnutý přepínač `mouse`. Takže myš označuje bloky přímo pro editor Joe, nikoli pro clipboard. Editor Joe neumí s clipboardem přímo pracovat (to by musel být slinkovaný s knihovnami X window systému, což po něm opravdu nechceme). Blok editoru Joe lze nicméně zkopírovat do clipboardu pomocí zkratky `Ctrl-N` (k tomu se využije `Shift`. V takovém případě se do clipboardu dostane text označený a viděný v Xtermu bez toho, aby se o tomto označení editor Joe vůbec dozvěděl. Xterm mu totiž tuto aktivitu zatají. Pro vyzvrácení clipboardu do místa kurzoru je možné použít `Shift-`prostřední tlačítko myši. Samotné prostřední tlačítko myši nedělá nic (při kompilaci z originálních zdrojáků). Nebo kopíruje označený blok editoru Joe (při použití mé záplaty před kompilací).

Zkratka `Ctrl-X~` prožene vyznačený blok nebo celý soubor programem `vlna`, tj. doplní vlnky za neslabičné předložky. Dále zkratka `Ctrl-XH` opraví v naposledy vlo6en0m textu probl0my zp;soben0 Mistrem Janem Husem. Tyto problémy mají lidé, kteří se chvíli nedívají na monitor, píšou a posléze zjistí, že měli špatně přepnutou klávesnici. Mezi tyto lidi patřím i já. Pojem „naposledy vložený text“ zahrnuje text vložený na jediném řádku po předchozím pohybu kurzoru nebo po zahájení editování souboru. Pokud ovšem vyznačíme nějaký blok a pak použijeme `Ctrl-XH`, korekce proběhne ve vyznačeném bloku.

Editor je nastaven tak, že se nepokouší interpretovat kódování načteného textu (mohl by, ale tuto funkci jsem v konfiguraci vypnul). Uvnitř ISO-2 terminálu editor předpokládá kódování ISO-8859-2 a uvnitř UTF-8 terminálu předpokládá UTF-8 kódování. Je-li načten soubor v nesprávném kódování, poznáme to podle paznaků. V takovém případě je možno použít zkratku `Ctrl-XE`, která v ISO-2

terminálu zkonvertuje buffer z UTF-8 do ISO-8859-2 a naopak v UTF-8 terminálu zkonvertuje buffer z ISO-8859-2 do UTF-8.

Funkce vyvolané klávesovými zkratkami `Ctrl-N` (kopie clipboardu), `Ctrl-XH` (Hus) a `Ctrl-XE` (korekce kódování) volají skripty, které jsem si pro tyto účely vytvořil a uložil do adresáře `./joe` (a zveřejnil na [1]). Tyto skripty pracují různým způsobem v závislosti na nastaveném parametru `charmap` v locales. Tedy v závislosti na použitém terminálu (ISO-2 nebo UTF-8). Je potřeba vědět, že pokud přejdeme v editoru do „křížového“ kódování, tj. editujeme např. ISO-8859-2 kódovaný soubor v UTF-8 terminálu, pak uvedené skripty nebudou pracovat správně.

Tak jako dříve v Emacsu, jsem soustředil i v editoru Joe TEXování na klávesu `F7`. Pomocí `Ctrl-F7` zahajuji proces `texloop` v nově založeném okně editoru. Editor si pamatuje posledně vložený příkaz (i příkazy starší), takže typicky nemusím vkládat celý příkaz `texloop` znovu. Poté mohu okno s procesem `texloop` skrýt (`F6`, `Ctrl-KI`) a mohu TEXovat pomocí `F7`. Skoro okamžitě po zmáčknutí této klávesy se mi obnoví zobrazení ve vedlejším prohlížeči PDF nebo DVI výstupu. Přitom (na rozdíl od Emacsu) okno s výpisem TEXu neobtěžuje, když nechci, ale mohu se tam podívat (`Ctrl-KI`).

Strasti a slasti s terminálem Xterm

Moje pracovní prostředí na počítači sestává z plochy Xserveru a na ní se vesměs rozprostírá spíše více než méně Xtermů. Klávesnice je implicitně anglická. Opravdu si nedovedu představit práci s příkazovým řádkem v české klávesnici. Jedna pracovní plocha Xserveru typicky nestačí, ovšem manager `fvwm` mi nabízí přepínat mezi devíti různými plochami. K tomu slouží klávesy `Ctrl-šipka`. To je důvod, proč jsem tyto sekvence nekonfiguroval jako klávesové zkratky pro editor. Občas na některém z devíti ploch je zapnuté i něco jiného než Xterm (například prohlížedlo www stránek, prohlížedlo PDF souborů atd.). Počet grafických programů ale nikdy nepřekročí množství Xtermů na pracovních plochách.

Uživatele počítačů se z mého pohledu dělí do dvou skupin, podle toho, jak přistoupí například k úloze „chci prohlédnout dokument v souboru `cosi.pdf` v adresáři `kdesi`“. První skupina spustí někde z nabídky window manageru program `xpdf` nebo něco podobného, pak tam v nabídce najde funkci „otevřít soubor“ a pak dlouze a podle abecedy hledá v periskopickém výpisu adresářů a souborů (s typicky zoufale malým průzorem) ten správný adresář a v něm ten správný soubor, pak klikne, a úloha je vyřešena. Druhá skupina vleze do Xtermu, v něm do příkazového řádku napíše „`cd kdesi Enter`“ (přitom velmi často využívá doplňování dlouhých názvů tabelátorem) a konečně napíše „`xpdf cosi.pdf`“. Nebo to urychlí a rovnou píše „`xpdf kdesi/cosi.pdf`“. Přiznám se, že já patřím do té druhé skupiny uživatelů. Proto potřebuji tolik Xtermů. Nutí-li mi někdo způsob

práce první skupiny, jsem velmi nešťastný zejména z periskopických hledáčků obsahujících výpisy souborů. Pravda, existuje ještě třetí skupina uživatelů, která typicky neví, co to je PDF, neví, co to je soubor, co to je adresář a používá jen metodu „klikání na obrázky“. Tato skupina využívá též toho, že window manager má jasno, který soubor otevřít kterým programem. Tato skupina uživatelů je mi ovšem už natolik myšlenkově vzdálená, že ji do svých úvah dále nebudu zahrnovat.

Pokud potřebuji během své práce výjimečně přepnout do české klávesnice, pak pouze v jednom zvoleném Xtermu, nikoli pro všechny X-ové aplikace globálně. Klávesnici tedy přepínám Xtermem. Používám k tomu konfiguraci Xtermu vytvořenou už velmi dávno Liborem Škarvadou, Petrem Mejzlíkem a Janem Kasprzakem. Pozor, vývojáři Xtermu v nejnovějších verzích cosi pokonili a přestala fungovat mrtvá klávesa. Je tedy potřeba mít verzi Xtermu nejvýše 256.

Přepnutí klávesnice tam i zpět se provede klávesou `Pause`. Při zapnuté České klávesnici kurzor bliká, při vypnuté je kurzor v klidu. Tím lze poznat stav klávesnice v každém Xtermu. Klávesou `Ctrl-Pause` změním implicitní černé pozadí Xtermu na bílé. Rád koukám do černé barvy, ale když je vedle rozsvícený bílý prohlížeč, je kontrast černá-bílá příliš unavující pro oči a v takovém případě přepnu Xterm na bílou.

Popsaná konfigurace Xtermu je pro kódování ISO-8859-2 v souboru `XTerm` a pro kódování UTF-8 v souboru `UXTerm`. Definici české klávesnice jsem v souboru `UXTerm` musel kompletně přepsat. Tyto soubory si můžete stáhnout z [1] a umístit do svého systému. Kam, to je závislé na systému. Já je mám typicky uloženy v `/usr/share/X11/app-defaults/`. Místo plných názvů fontů jsou v souboru `XTerm` uvedeny aliasy, které je potřeba nastavit v souboru `fonts.alias`. Jak, to také závisí na systému.

Na stránce [1] je též podrobný popis instalace Xtermu. Bohužel mám špatné zkušenosti s přenosem češtiny v clipboardu, pokud se použije v ISO-2 locales Xterm, který je kompilován i pro UTF-8. Mám proto v systému binárky dvě. Jednu, která je kompilovaná jen pro jednobytové kódování a druhá, která je kompilovaná pro UTF-8. Pak přenos v clipboardu probíhá spolehlivě i mezi různě kódovanými aplikacemi. Interní obsah clipboardu je vždy UTF-8.

V popisu instalace je zmíněna i možnost mít v systému (třeba na ploše vedle sebe) oba Xtermy (ISO-2 i UTF-8). Implicitně mám nastaveno v systému `LC_CTYPE=cs_CZ.ISO8859-2` a pro UTF-8 kódovaný Xterm přepínám do `LC_CTYPE=cs_CZ.UTF-8`. Ostatní proměnné z locales (včetně proměnné `LANG`) nejsou nastaveny, protože samozřejmě nechci, aby na mě příkazový řádek štěkal česky. To snad nechce žádný soudně uvažující uživatel.

UTF-8 kódovaný csplain

Formát `csplain` používám implicitně v kódování ISO-8859-2. Výhody přechodu na kódování UTF-8 nejsou v případě `csplainu` vůbec zřetelné. Znaky mimo znakovou sadu ISO-8859-2 stejně musíme individuálně ošetřit makry v TEXu. Takže mě nic nenutí explicitně přecházet na jiné kódování.

Někdy se ovšem stane, že zákazník dodává soubory v UTF-8 a my je chceme bez potřeby neustálého překódování zpracovávat v csplainu. Pak je možné samozřejmě tyto soubory zpracovávat v UTF-8 kódovaném terminálu. Místo příkazu `csplain` pak použiji příkaz `ucsplain` (respektive `pdfucsplain`). To je formát, který překódovává vstup z UTF-8 do ISO-8859-2 na úrovni input procesoru TEXu a využívá k tomu encTEX. Výstup na terminál, do logu a do `\write` souborů se vrací zase v kódování UTF-8. Formát vygenerujeme takto:

```
pdftex -ini -enc \\let\\enc=u \\input csplain.ini
mv csplain.fmt ucsplain.fmt
mv csplain.log ucsplain.log
pdftex -jobname pdfcsplain -ini -enc \\let\\enc=u
                                    \\input csplain.ini
mv pdfcsplain.fmt pdfucsplain.fmt
mv pdfcsplain.log pdfucsplain.log
su
cp ucsplain.* pdfucsplain.* /var/lib/texmf/web2c/pdftex/
texhash
cd /usr/bin
ln -s pdftex ucsplain
ln -s pdftex pdfucsplain
```

Nyní jsou všem uživatelům v systému přístupné příkazy `ucsplain` a `pdfucsplain`, kterými mohou zpracovávat zdrojové soubory kódované v UTF-8.

Odkazy

[1] `http://petr.olsak.net/texloop.html`
[2] `http://joe-editor.sourceforge.net/`

Summary: TEX in the simple Unix environment

By debugging a TEX document it is necessary many times repeatedly to run TEX, to look for the result in DVI or PDF file, to gander the TEX output on the terminal, or eventually to have a look in the log-file, and all that action to repeat. In the paper it is show, how this work is made by author. The process

‘editor-TEX-look’ is supported by simple Unix tools: bash script `texloop`, created by author for these purposes, Xterm terminal and a simple editor, which is able to link to the shortcut key the activation of a system command. The reader could be inspired with the solution and to adapt these tools to his/her own needs. In the paper the function of the `texloop` script is described, and further the longstanding evolution of the author’s relation to text editors is informal elaborated and finally a configuration of Xterm terminal, suitable for the czech environment with both ISO-8859-2 and UTF-8 encoding is introduced. For UTF-8 encoding the TEX format `csplain` is generated at the end of the paper.

Key words: script `texloop`, Unix, editor Joe

Zpráva z konference 5th ConTEXt meeting 2011

Tomáš Hála

(pokračování z minulého čísla)

V příspěvku *MetaPost, how we adapt* ukázal Hans Hagen (Nizozemí) nové vlastnosti knihovny MPlib, které byly implementovány v průběhu posledního roku. Mezi ně patří i odstanění některých přísných omezení, která vedou k snadnějšímu užití MetaPost pro kreslení grafů a vizualizaci dat. Diskutoval také otázku zavedení tzv. jmenných prostorů.

Luigi Scarso (Itálie) se zabývá podporou grafiky, a tak i jeho vystoupení *Extending* ConTEXt *with GraphicMagick: When bitmaps beats vectors* bylo laděno tímto směrem. Po seznámení posluchačů s knihovnou Wand pro GraphicsMagick a vlastnostmi rozhraní API, které tato knihovna implementuje, následoval popis knihovny SWIG, napsané v jazyce C a používané pro spojení Lua a GraphicsMagick. Celá řada příkladů ilustrovala různé možnosti tohoto řešení, například programovatelné úpravy barevnosti a kontrastu při zpracování fotografií či konverze do stupňů šedi nebo monochromatického obrazu.

Dále představil program SimpleCFDG, jehož cílem je generovat obrázek na základě instrukcí zvláštního programovacího jazyka. Vzhledem k tomu, že použité prostředí umožňuje rekurzivní volání funkcí, byla tato technologie prezentována mj. na fraktálových obrazcích.

Alan Braslau (Francie) v příspěvku *Graphics, in particular plotting data* předvedl kreslení grafů pomocí MetaPostu. Po rozboru nástrojů vhodných použití a měrných soustav předvedl příklady zaměřené na základní organizaci grafu, správu popisků os a pomocných linií, kreslení křivek, vynesení bodů simulující data, volbu tvarů pro vyznačení bodů, automatickou volbu barvy křivky v grafu s více křivkami, různé způsoby prezentace trojrozměrných dat.

Jean-Michel Hufflen (Francie), který se dlouhobě zabývá podporou sazby bibliografických citací, se zaměřil na aktuální technologie použitelé v ConTEXtu a LuaTEXu. Vystoupení *Bibliography tools and* ConTEXt*/LuaTEX* bylo orientováno na využití značkovajího jazyka XML.

Thomas Schmitz (Německo) si připravil tutoriál s názvem *Presentations in XML with the simpleslides module*, v němž využil možnosti vstupu v XML pro

překlad ConTeXtem, a předvedl zápisy jak přímo v jazyce XML, tak příkazy ConTeXtu určenými pro popis XML včetně kompletního příkladu.

O dalších vybraných vystoupeních již jen stručně.

Hans Hagen, *Sorting registers*, ukázal specifický požadavek na řazení v korejštině a zamýšlel se nad řazením ve vícejazyčném prostředí.

Thomas Schmitz, *XML in* ConTeXt *MkIV (for beginners)*, připravil krátký kurs využití XML pro začátečníky.

Jano Kula (Česká republika), ConTeXt *XML processing*, rovněž přispěl k problematice zpracování XML a převedl ukázky z oblasti přípravy materiálů pro Mezinárodní filmový festival.

Mojca Miklavec, *State of the standalone distribution*, pohovořila o tom, co je ještě potřeba dokončit pro samostatnou distribuci ConTeXtu.

Možnosti elektronických knih diskutoval Hans Hagen, *EPub generation*. Mimo jiné si položil otázku, zda ConTeXt může generovat výstup v HTML, a předvedl základní návrh řešení.

Taco Hoekwater, *Publishing* ConTeXt *modules revisited*, učinil významné úpravy v oblasti umístění modulů na webu `contextgarden.net`, a ukázal jak použít nový systém pro nahrávání a stahování modulů.

Hans Hagen (*MathML, an update*) uvedl aktualizaci MathML pro Unicode.

Ulrik Vieth (Německo), *OpenType math fonts*, prověřil možnosti matematických fontů Cambria Math, Asana Math, XITS Math a Latin Modern Math a srovnal jejich možnosti.

John Haltiwanger (Nizozemí), *The Markdown module and the beauty of pre-formats*, se zaměřil na integraci nového modulu `m-markdown` Hanse Hagena v ConTeXtu, a to v souvislosti s požadovanými různými výstupními formáty při sazbě.

Bohatý odborný program byl doplněn o čtvrteční exkurzi do blízkého okolí. Přesunuli jsme se do malého městečka Canne, jež je rozděleno Albertovým kanálem na dvě části. V severní části (Opcanne) se nacházejí bohatě rozvětvené vápencové jeskyně, které vznikly těžbou této suroviny. Strukturu vyhloubených jeskyní přibližuje ručně kreslený nákres, viz foto na další straně.

V minulosti posloužily i jako úkryty, současné využití je různorodé – nachází se zde společenský a koncertní sál, pěstírna žampiónů či sklady vína.

Exkurze potom pokračovala návštěvou několikapatrové podzemní pevnosti Eben-Emael, jejíž rychlé a nečekané dobytí německým letectvem 10. května vtáhlo Belgii do druhé světové války.

Pevnost byla vybudována během třicátých let 20. století, aby bránila případnému postupu německé armády. Její umístění a vybavení umožňovalo chránit mosty přes řeku Mázu. Německá armáda použila k dobytí tažné kluzáky, které mohly přistát na malém prostoru, např. na fotbalovém hřišti, vybudovaném nad pevností. Pět set německých výsadkářů dobylo po 33 hodinách pevnost s dvěma tisíci muži.

Osmnáct dní po dobití pevnosti Belgie kapitulovala.

Závěrečná zastávka exkurze se uskutečnila na hranicích Belgie a Nizozemí v Neercanne, na zámku s barokní zahradou upravenou do původní podoby a s výhledem do údolí Jeker.

Večer po exkurzi se účastníci přenesli v čase o pět století nazpět a vyslechli ukázky středověké hudby v provedení Maaike Boekhout (Nizozemí, loutna) a Reginy Albanez (Brazílie, viola da gamba). S ukázkami hudby spojila Mari Voipio (Finsko), ve vlastnoručně zhotoveném dobovém kroji, i výuku středověkých tanců. Zúčastnili se však jen odvážnější.

Páteční večer byl věnován prvnímu valnému a současně volebnímu shromáždění ConTeXt Group, což je zájmové sdružení podporující ConTeXt. Prezidentem se stal Taco Hoekwater, jeho zástupcem Mojca Miklavec, pokladníkem Adelheid Grob, tajemníkem Willi Egger.

V pátek konferenci navštívila také Siep Kroonenberg (Nizozemí) a představila svůj nový návrh webových stránek `contextgarden.net`, který byl přijat a posléze i užit.

Plné texty vybraných příspěvků a dále abstrakty zbývajících lze nalézt ve sborníku, který vyšel ve dvou vydáních: nejprve v tzv. preprintu těsně před konáním konference, a poté, s jistým časovým odstupem, ve finální podobě. Obě vydání jsou k dispozici u každého z účastníků konference.

Poděkování

Závěrem je mou milou povinností poděkovat Československému sdružení uživatelů TeXu za finanční podporu mé účasti na této akci.

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
Počet výtisků:	400
Uzávěrka:	15. 7. 2012
Odpovědný redaktor:	Zdeněk Wagner
Redakční rada:	Zdeněk Wagner (šéfredaktor), Ján Buša, Jiří Demel, Tomáš Hála, Jaromír Kuben, Michal Růžička, Jiří Rybička, Petr Sojka, Pavel Stříž, Jan Šustek
Technický redaktor:	Tomáš Hála
Tisk a distribuce:	KONVOJ, spol. s r. o., Berkova 22, 612 00 Brno, tel. +420 541 245 548
Adresa:	CSTUG, c/o FEL ČVUT, Technická 2, 166 27 Praha 6
E-mail:	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`

webový server sdružení:
`http://www.cstug.cz`

CONTENTS