TUGboat

Volume 36, Number 2 / 2015
TUG 2015 Conference Proceedings

TEX Users Group

TUGboat (ISSN 0896-3207) is published by the TEX Users Group.

Memberships and Subscriptions

2015 dues for individual members are as follows:

- Regular members: $105.
- Special rate: $75.

The special rate is available to students, seniors, and citizens of countries with modest economies, as detailed on our web site. Also, anyone joining or renewing before March 31 receives a $20 discount.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site.

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate for 2015 is $110.

Institutional Membership

Institutional membership is primarily a means of showing continuing interest in and support for both TEX and the TEX Users Group. It also provides a discounted membership rate, site-wide electronic access, and other benefits. For further information, see `http://tug.org/instmem.html` or contact the TUG office.

Trademarks

Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is. The following list of trademarks which commonly appear in *TUGboat* should not be considered complete.

TEX is a trademark of American Mathematical Society.
METAFONT is a trademark of Addison-Wesley Inc.
PostScript is a trademark of Adobe Systems, Inc.

[printing date: September 2015]

Printed in U.S.A.

Addresses

TEX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Telephone

+1 503 223-9994

Fax

+1 815 301-3568

Web

`http://tug.org/`
`http://tug.org/TUGboat/`

Electronic Mail
(Internet)

General correspondence, membership, subscriptions:
`office@tug.org`

Submissions to *TUGboat*, letters to the Editor:
`TUGboat@tug.org`

Technical support for TEX users:
`support@tug.org`

Contact the Board of Directors:
`board@tug.org`

2015 Conference Proceedings

TEX Users Group
Thirty-sixth Annual Meeting
Darmstadt, Germany
July 20–22, 2015

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
TUGboat Editor Barbara Beeton
Proceedings Editor Karl Berry

Volume 36, Number 2 • 2015
Portland • Oregon • U.S.A.

TUG 2015 — Darmstadt, Germany

July 20–22, 2015 ▪ Welcome Hotel

Sponsors

TEX Users Group ▪ DANTE e.V. ▪ River Valley Technologies — UK
with special assistance from individual contributors. *Thanks to all!*

Conference committee

Karl Berry ▪ Klaus Höppner ▪ Robin Laakso ▪ Volker RW Schaa ▪ Joachim Schrod

Bursary committee

Taco Hoekwater, chair ▪ Jana Chlebikova ▪ Kaja Christiansen ▪ Bogusław Jackowski ▪ Alan Wetmore

Participants

Leyla Akhmadeyeva, Bashkir State Medical Univ.
Pavneet Arora, Bolton, ON
Kaveh Bazargan, River Valley Technologies, UK
Stefan Bedacht, TU Darmstadt
Barbara Beeton, AMS
Nelson Beebe, University of Utah
Denis Bitouzé, Université du Littoral Côte d'Opale
Johannes Braams, Zoetermeer, Netherlands
Gyöngyi Bujdosó, University of Debrecen
David Carlisle, LaTeX3 Project
Jennifer Claudio, Synopsys Outreach Foundation
Julien Cretel, University College Cork
Rajagopal CV, River Valley Technologies, India
Christine Detig, Net & Publication Cons. GmbH
Michael Doob, University of Manitoba
Olaf Drümmer, callas software GmbH
Graeme Duffin, Huddersfield, UK
Dominik Fischer, TU Darmstadt
Ulrike Fischer, Mönchengladbach, Germany
Yukitoshi Fujimura, Ichikawa-shi, Japan
Deimantas Galčius, VTEX
Roland Geiger, Leipzig, Germany
Paul Gessler, Milwaukee, WI
Steve Grathwohl, Duke University Press
Gary Gray, State College, PA
Enrico Gregorio, Università de Verona
Hans Hagen, Pragma ADE
Tom Hejda, Czech Technical University in Prague
Klaus Höppner, TUG and DANTE e.V
Bogusław Jackowski, Gdańsk, Poland
Timm Knape, Wehrheim, Germany
Harald König, Balingen, Germany
Jonathan Komar, ITH icoserve
Stefan Kottwitz, Lufthansa Industry Solutions
Reinhard Kotucha, Hannover, Germany
Siep Kroonenberg, Groningen, Netherlands
Sebastian Krüger, Berlin, Germany
Yusuke Kuroki, Yokohama, Japan
Dag Langmyhr, University of Oslo
Bruno Le Floch, LaTeX3 Project
Manfred Lotz, DANTE e.V.
Jerzy Ludwichowski, Toruń, Poland
Henri Menke, Leinfelden-Echterdingen, Germany
Lothar Meyer-Lerbs, Bremen, Germany
Frank Mittelbach, LaTeX3 Project
Ross Moore, Macquarie University
Gerd Neugebauer, Groß-Gerau, Germany
Heiko Oberdiek, Sasbach, Germany
Petr Olšák, Czech Technical University in Prague
Steve Peter, TUG
Susanne Raab, ECAP, Uni Erlangen
Arthur Reutenauer, Royal Opera House, London
Will Robertson, University of Adelaide
Petra Rübe-Pugliese, CTAN
Volker RW Schaa, Darmstadt, Germany
Joachim Schrod, Net & Publication Cons. GmbH
Martin Schröder, Duisburg, Germany
Torsten Schuetze, Talheim, Germany
Herbert Schulz, Naperville, IL
Peter Selinger, Dalhousie University
Keiichiro Shikano, Tokyo, Japan
Martin Sievers, DANTE e.V.
Matthew Skala, Copenhagen, Denmark
Linas Stonys, VTEX
Piotr Strzelczyk, Gdańsk, Poland
S.K. Venkatesan, TNQ Software
Boris Veytsman, George Mason University
Ulrik Vieth, Stuttgart, Germany
Herbert Voß, DANTE e.V.
Alan Wetmore, US Army Research Laboratory
Joseph Wright, Northampton, UK

TUG 2015 — program and information

Sunday, July 19, 3 pm: **walking tour** of Mathildenhoehe and Rosengarten, duration about 2 hr, with our Klaus Höppner as guide.

Sunday, July 19, 7 pm: **informal opening gathering**, LaLucha, Schleiermacher Str. 10-12 (self-pay basis).

Monday, July 20: **post-session TeXShop workshop**, Herb Schulz.

Monday	8:00 am	*registration*	
July 20	8:50 am	Steve Peter, TUG	*Opening*
	9:00 am	Ross Moore	*Semantic enrichment of mathematics using 'active comments'*
	9:35 am	Olaf Drümmer	*PDF/UA — what it is, how users can benefit from it, and how to get it right*
	10:10 am	Ross Moore and Peter Selinger	*Using* `pdfx.sty` *for producing validating PDF documents*
	10:45 am	*break*	
	11:00 am	Joseph Wright	*XeTeX and LuaTeX: Getting Unicode data into the right places*
	11:35 am	Will Robertson	*Reconciling* `unicode-math` *with LaTeX2ε mathematics*
	12:10 pm	Bogusław Jackowski, Piotr Strzelczyk, and Piotr Pianowski	*All the characters we need*
	12:45 pm	*lunch*	
	2:00 pm	Bogusław Jackowski, Piotr Strzelczyk, and Piotr Pianowski	*Six GUST e-foundry math fonts and what next?*
	2:35 pm	Frank Mittelbach	*Twenty one is only half the truth*
	3:10 pm	Hans Hagen	*What if . . .*
	3:45 pm	*break*	
	4:00 pm	Joseph Wright	*State of the (UK-)TeX FAQ*
	4:35 pm	CTAN team	*State of CTAN*
	4:45 pm	Barbara Beeton, Volker RW Schaa, Joachim Schrod	*In memoriam: Pierre MacKay, Richard Southall, Hermann Zapf, Thomas Koch*
Tuesday	8:55 am	*announcements*	
July 21	9:00 pm	Pavneet Arora	*Fluss: A flow leak monitoring system*
	9:35 am	Tom Hejda	*Preparing LaTeX classes/templates for journal articles and university theses*
	10:10 am	Boris Veytsman and Michael Cohen	*A new multibibliography package:* `nmbib`
	10:45 am	*break*	
	11:00 am	Boris Veytsman and Leyla Akhmadeeva	*Trilingual templates for an educational institute in Bashkortostan, Russia*
	11:35 am	Paul Gessler	*Pretty-printing Git commit history graphs with PGF/TikZ*
	12:10 am	q&a, TUG meeting	
	≈12:30 pm	*lunch*	
	2:30 pm	*Messel Pit excursion*	
Wednesday	8:55 am	*announcements*	
July 22	9:00 am	Kaveh Bazargan and Jagath AR	*TeX — After 35 years, still the best solution for modern publishing*
	9:35 am	Joachim Schrod	*DocCenter: TeXing 11 million documents a year*
	10:10 am	S.K. Venkatesan	*A proposal to construct pagination as a three-step cookie-cutter process*
	10:45 am	*break*	
	11:00 am	Joseph Wright	*Through the* `\parshape`*, and what Joseph found there*
	11:35 am	Julien Cretel	*Functional data structures in TeX*
	12:10 pm	Hans Hagen	*When to stop . . .*
	12:45 pm	*lunch*	
	1:45 pm	*group photo*	
	2:00 pm	Boris Veytsman	*TeX and controlled access to information*
	2:35 pm	Enrico Gregorio	*Recollections of a spurious space catcher*
	3:10 pm	q&a	
	7:00 pm	*banquet*	at Cafe Rodenstein, in the museum building.

Torsten Schuetze, Siep Kroonenberg, Herb Schulz, and Keiichiro Shikano

Jonathan Komar, Ross Moore, and Olaf Drümmer

Deimantas Galčius, Jagath AR, Enrico Gregorio, and Linas Stonys

SK Venkatesan, Jagath AR, Herb Schulz, and Keiichiro Shikano

TEX Lions: Jerzy Ludwichowski (GUST), Hans Hagen (NTG), and Volker Schaa (DANTE)

Heiko Oberdiek, Nelson Beebe, and Graeme Duffin

Dominik Fischer, Paul Gessler, and Tom Hejda

Excursion to Messel Pit

A well at Messel Pit with 10^8 year-old water; smells a bit like sulfur but tastes good

Stefan Kottwitz, Deimantas Galčius, and Linas Stonys

City tour with Klaus

Darmstadt at night

TUG Board: Pavneet Arora, Michael Doob, Barbara Beeton, Arthur Reutenauer, Klaus Höppner, Boris Veytsman, and Steve Grathwohl

CTAN maintainers: Manfred Lotz, Joachim Schrod, Gerd Neugebauer, and Petra Rübe-Pugliese

LaTeX developers: Frank Mittelbach, Johannes Braams, David Carlisle, Will Robertson, Joseph Wright, and Bruno Le Floch

Joachim Schrod presents drawings by Duane Bibby to Klaus Höppner for the excellent conference organization and . . .

to Ulrike Fischer for her tremendous efforts supporting TeX users

At the banquet

Front row: Keiichiro Shikano, Volker Schaa, Yusuke Kuroki, Ulrike Fischer, Lothar Meyer-Lerbs, Alan Wetmore, Leila Akhmadeeva, Boris Veytsman, Steve Peter, Jennifer Claudio, Gyöngyi Bujdosó, Tomasz Łuczak, Enrico Gregorio, Will Robertson, Jagath AR, Reinhard Kotucha
2nd row: Pavneet Arora, Frank Mittelbach, Jonathan Komar, Dominik Fischer, Steve Grathwohl, Susanne Raab, Matthew Skala, Harald König, Roland Geiger, Ulrik Vieth, Ross Moore, Barbara Beeton, Johannes Braams, Petr Olšák, Gerd Neugebauer, Christine Detig, Manfred Lotz, Joachim Schrod, Nelson Beebe, Michael Doob, Denis Bitouzé, Siep Kroonenberg, Kaveh Bazargan, Deimantas Galčius
3rd row: Jerzy Ludwichowski, Hans Hagen, Sebastian Krüger, Stefan Bedacht, Dag Langmyhr, Timm Knape, Bogusław Jackowski, Piotr Strzelczyk, Gary Gray, Klaus Höppner, Herbert Schulz, Heiko Oberdiek, Tom Hejda, Joseph Wright, Paul Gessler, Henri Menke, Rajagopal CV, SK Venkatesan, Herbert Voß, Stefan Kottwitz, Petra Rübe-Pugliese, Bruno Le Floch, Arthur Reutenauer, Torsten Schuetze, Graeme Duffin
4th row: David Carlisle, Martin Schröder, Yukitoshi Fujimura, Julien Cretel, Linas Stonys

Photos courtesy of Alan Wetmore, Pavneet Arora, and Reinhard Kotucha

Menu

Essen, wie gedruckt!
Food, in Typographer's Terms

3erlei Vorspeisenteller
3 kinds of appetizers

ZWIEBELFISCH	• Pulposalat mit Orange und roten Zwiebeln • Salad of pulpo (octopus) with orange and red onions
EIERKUCHEN	• Muskatcrêpe mit Zitronenfrischkäsecreme, Staudensellerie, Tomate • Nutmeg crêpe with lemon cream cheese, celery, and tomato
SPECK & SCHUSTERJUNGE	• Längliches Brötchen mit Pancetta und Gemüsesalat • Sandwich with pancetta and vegetable salad

Hauptspeise nach TYPOGRAPHISCHEM MASS
Main dish according to TYPOGRAPHIC MEASURE

KONKORDANZ 5x5	• Lasagne mit Sugo vom Rind und Schwein mit gegrilltem Gemüse oder • Gemüselasagne mit Mozzarella und Tomaten (vegetarisch) • Lasagne with beef and pork sugo and grilled vegetables or • Vegetable lasagne with mozzarella and tomatoes (vegetarian)

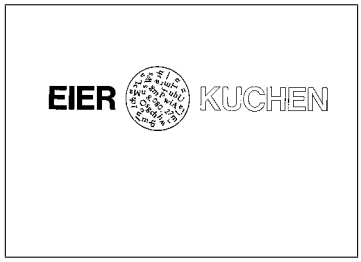
Nachspeisen
Desserts

MAGER, HALBFETT	• Käseplatte • Cheese platter
EXTRAFETT	• Panna Cotta mit roten Beeren • Panna cotta with red berries

---

Zwiebelfisch
- Zwiebelfische sind Buchstaben einer bestimmten Schrift, die aus Unachtsamkeit in einem anderen Schriftkasten abgelegt wurden.
  Im Satz erscheinender Buchstabe anderen Schriftcharakters.

Printer's Pie
- Letters sorted into the wrong box of a type case (different typeface, size, etc.).
  A letter set in a wild typeface on a typeset page.



Eierkuchen
- Bezeichnung für einen auseinandergefallenen Satz: ein *kleines* Missgeschick – ein mit viel Mühe gesetzter Schriftsatz, der in sich zusammenfällt und in seine Bestandteile auflöst. Der Eierkuchen ist fertig.

(fall into) Pie
- A *small* mishap and a page typeset with a lot of effort collapses into its constituents and dissolves.
  The pie is ready.

TUG2015 — Darmstadt, Germany

Menu

Speck

Speck

- Bezeichnung der Setzer für die leeren Räume in der Satzform, so z. B. die leeren Rückseiten des Titelblattes, des Widmungsblattes u. ä., die offenen Seiten in Kapitelanfängen usw. Diese Räume wurden dem Setzer wie eine normale Satzarbeit bezahlt.

?

- Typesetter's term for the empty spaces on pages, empty backside of title or dedication pages, the open page in chapter beginnings, etc. These spaces have been paid like a normal typeset page.

Schusterjunge

Schusterjunge

- Ein Schusterjunge ist die erste Zeile eines Absatzes, wenn sie allein am Ende einer Spalte oder einer fertigen umbrochenen Buchseite steht.

Orphan

- A paragraph-opening line that appears by itself at the bottom of a page or column.

Fleisch
Nach Typographischem Maß

5 Konkordanz

Fleisch

- Fleisch nennt man den leeren Raum, der den Buchstaben umgibt. Der das Buchstabenbild umgebende Teil der Oberfläche einer Drucktype, der tiefer liegt als das Buchstabenbild (siehe 1 im linken Bild).

shoulder/neck

- conflicting definitions

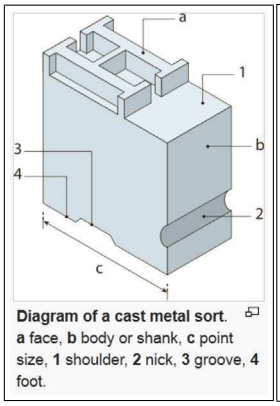
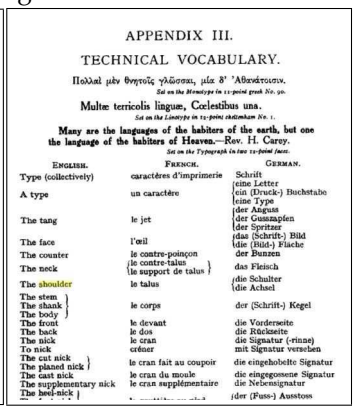


Diagram of a cast metal sort. a face, b body or shank, c point size, 1 shoulder, 2 nick, 3 groove, 4 foot.


APPENDIX III.

TECHNICAL VOCABULARY.

Πολλαὶ μὲν θνητοῖς γλῶσσαι, μία δ' Ἀθανάτοιυν.

Multæ terricolis linguæ, Cœlestibus una.

Many are the languages of the habiters of the earth, but one the language of the habiters of Heaven.—Rev. H. Carey.

English.	French.	German.
Type (collectively)	caractères d'imprimerie	Schrift
A type	un caractère	eine Letter / ein (Druck-) Buchstabe / eine Type
The tang	le jet	der Anguss / der Gussnapfen / der Spritzer
The face	l'œil	das (Schrift-) Bild / die (Bild-) Fläche
The counter	le contre-poinçon	der Punzen
The neck	le contre-talus / le support de talus	das Fleisch
The shoulder	le talus	die Schulter / die Achsel
The stem / The shank / The body	le corps	der (Schrift-) Kegel
The front	le devant	die Vorderseite
The back	le dos	die Rückseite
The nick	le cran	die Signatur (-rinne)
To nick	créner	mit Signatur versehen
The cut nick / The planed nick	le cran fait au coupoir	die eingehobelte Signatur
The cast nick	le cran du moule	die eingegossene Signatur
The supplementary nick	le cran supplémentaire	die Nebensignatur
The heel-nick		der (Fuss-) Ausstoss

5 Konkordanz

Konkordanz

- Typographisches Maß
  48 Didot ⇒ 4 Cicero ⇒ 1 Konkordanz
  5 Konkordanz ⇒ ∼90 mm

4-em quad

- Typographic measure
  48 Didot ⇒ 4 Cicero ⇒ 4 em quad
  5 Konkordanz ⇒ ∼3.5 in

Mager
Halbfett

Extrafett

Mager
Halbfett
Extrafett

- Abbildung (Schnitt) des jeweiligen Schriftschnittes einer bestimmten Schrift

Light
Semibold
Extra Bold

- Font weights

TUG 2015 conference report

Stefan Kottwitz

TUG 2015 — the day before

The TUG 2015 conference took place in Darmstadt, Germany, from 20th to 22nd of July. This was the 36th annual meeting of the international TEX Users Group. DANTE e.V. (`http://www.dante.de`) sponsored the conference fee for its members with 50 Euro for each attendee. I'm sure this great sponsorship helped many TEX friends to come.

By way of introduction, I work for Lufthansa Industry Solutions (`lufthansa-industry-solutions.com`), developing and implementing networks for cruise ships. I planned to visit the conference privately. When I told my project team that I will leave for three days, they asked me why. Conference? TEX? What is this? I explained what TEX is and how I used it in my work. I use TEX as macro language for creating thousands of lines of switch configurations. And I create graphics visualizing the physical and logical structure of networks using TEX coding, specifically Ti*k*Z — what other people point and click with Visio. That convinced my boss to consider the TEX meeting as training, so the company covered my travel costs. He expects that our work in designing and documenting will benefit, and this is true.

I arrived on Sunday right before the conference. There was an informal gathering at 7 pm in a restaurant, so I walked there. The restaurant was pretty full of TEX friends, seems like almost everybody was already there. That was a great occasion to meet people again, some I have not seen since 2011, when I attended the TUG meeting in Kerala, India. One such was Kaveh Bazargan, who has been elected to be the next TUG president. His plans are very interesting, as he is interested in boosting online presence, attracting new users, and attracting publishers to get things funded. Of course I met many DANTE members, whom I saw last in Stralsund earlier this year at the DANTE meeting. I also had interesting discussions, such as about `tex4ht`, with two men from VTEX (`http://www.vtex.lt`), a LATEX-based publishing company based in Vilnius, Lithuania. I met Reinhard Kotucha again, who is a regular at TEX and DANTE meetings for a long time. When you arrive in a restaurant and don't know most of the people yet, it's good to see a familiar face. As usual, Reinhard takes photos of the conference and the surrounding world.

Editor's note: Originally posted at `latex-community.org` by the author; edited for *TUGboat*, with permission.

Finally, only a group of DANTE people remained. About 11 pm it was time to return to the hotel.

TUG 2015 — day 1

About 9 am, the current TUG president, Steve Peter, opened the conference with some introductory words.

The topics for today:

- PDF: enriching it and making it accessible
- Unicode: getting it into TEX
- Past, present and future of TEX, LATEX and fonts
- News and announcements

Let's take a look at the presentations.

Ross Moore gave the first talk. He spoke about semantic enrichment of mathematics. At first, he demonstrated the already available possibilities to add PDF tooltips to text and math. Tooltips mean text boxes popping up when you move over it with the mouse or cursor. While they won't be printed, they add value to electronic documents. The reader is able to access further information, which is otherwise hidden to stay focused. Constructing math expressions with embedded semantics requires a difficult syntax though.

Ross introduced a new package called `mathsem`. It offers a way to provide the semantic meaning separate from a math formula. Here's an example:

```
\(
%$semantics
% x $ variable
% f $ function
% \Re $ real part of
%$endsemantics
y = f(x)
\)
```

The characters in the formula now get their own tooltip. As you can see, you can define semantics for macros. This is in fact recommended: define a macro with a name explaining the meaning, rather than how it's typeset, and add semantic information to the macro. The syntax is:
`%` *token* `$` *semantics end-of-line*

It's implemented by hacking the catcodes of the comment symbol `%` and end-of line, and then hooking in. This way has the benefit that even without the package, everything just works, since the additions are hidden in comments and look like code annotations, thus still useful in themselves.

To ease the work, `mathsem` provides a command `\DeclareMathSemantics` for repeated use, setting up default tooltips for symbols and macros. The tooltips can be also be used for a screen reader, meaning for vocalization by assistive tools. Ross

showed several examples, and discussed the application to have spoken words for math, in the context of PDF/UA documents.

Following this, Olaf Drümmer explained what PDF/UA means, in the following talk. UA stands for universal accessibility, which basically means that a document provides reasonable access to the PDF content. So, a visually impaired person could use a screen reader to get vocal information. It requires that all content (including images) be available as text. Other demands are reliable, defined order; embedded semantics; a logical structure; metadata; and not using encryption. PDF/UA is related to tagged PDF: a well tagged PDF can conform to PDF/UA. Olaf demonstrated screen reader software, a PDF/UA checker, and a visualizer tool.

Another talk by Ross Moore together with Peter Selinger gave us an update to new developments of the `pdfx` package, which helps in producing documents conforming to PDF/A standards. There are various PDF/A-x standards, and the new version of `pdfx` supports most of them. Furthermore, the new version adds support for LuaTEX.

Now we got a coffee break — a few minutes to fetch a coffee and to drink it, talking a bit; I wish it could be longer as I value such conversations during conferences.

Joseph Wright gave the first talk after the break. While the first session was about PDF, we now come to the topic of Unicode. He spoke about getting Unicode input into TEX and the challenges involved. For example, he explained the difficulties in uppercasing, lowercasing, titlecasing and casefolding. Details can give a developer headaches, I feel. There are a lot of differences to consider in various languages.

Then it was Will Robertson's turn. At the stage, he changed roles, grabbing his camera and said: "Everybody wave!" and took a picture of the audience. Then, he started with a retrospective about his development of the `unicode-math` package.

`unicode-math` allows us to switch math fonts as easily as switching text fonts. There are thousands of math glyphs in various fonts, each one with a LATEX name, but you can also simply use it as an input symbol. This can be done by code-auto-completion of the LATEX command to that symbol. This may increase readability, but not in all cases, such as when we have glyphs that are too similar. Thus, `unicode-math` gives direct access to a huge collection of symbols. A font with proper Unicode support is required, of course. Luckily, there are some.

LATEX authors commonly use fonts to convey a meaning. In Unicode mathematics, you keep the same font but choose a symbol with the desired meaning. There are a lot of spacing challenges because it is done differently in math compared to text.

Next, the GUST team discussed how characters for math fonts are chosen. That led into a talk about whether we really need new fonts, or if there's not enough demand. We heard about the TEX Gyre math fonts, described with their underlying scheme, and with variants of bold, italic, sans-serif, double-stroke and more. Requirements were discussed such as scaling factors for subscripts and superscripts, math kerning, glyph links, growing glyph chains. There are over 4200 glyphs in DejaVu Math alone.

But few companies produce OpenType math fonts. So, perhaps there's no commercial pressure for math fonts, or not enough demand. So, a future task is possibly not making just another font, but font variants, such as sans-serif variants for presentations, or bold/heavy variants for headings.

Frank Mittelbach then talked about history and current development of LATEX 2ε. Now, it's 21 years old. The policy of compatibility will now change to a policy of roll back–roll forward. Fixes and enhancements will be applied directly to the kernel. You can call the `latexrelease` package with a date as option, and it will change to be compatible to the version of that date. Also packages can adjust their code to releases via an `\IncludeInRelease` macro. There will be also patch releases which will not be roll-back points, in contrast to major releases.

All the fixes of `fixltx2e` are now in the kernel. ε-TEX support is now included out-of-the-box, along with fundamental support for XETEX and LuaTEX, a regression suite for testing with all formats. ε-TEX and XETEX passed the tests, while still there are many failures in running against LuaTEX, to be examined. Some improvements as well:

- `\emph` can now produce small caps or others
- `\textsubscript` is defined
- `\DeclareMathSizes` only takes points
- fewer fragile commands

The final speaker for the third session was Hans Hagen. His talk "What if ..." was more reflective. He looked at Unicode, which is a nice way to get rid of input encodings and font encodings, with easy transition thanks to existing UTF-8 support. And there are sophisticated casing, categories, and spacing. What if we had had it earlier? A lot of time was "wasted" struggling with encodings. However, Unicode may introduce challenges due to possibilities, cultural issues of symbols, and persistent compatibility errors. And, there are exceptions due to user demands.

He reflected on TEX's design, which is nice but

may be boring sometimes. The look and feel may not fit some purpose such as schoolbooks. Generally, nice fonts help reading, but of course there are different opinions on design.

Hans explained that we still have insufficient but persistent standards. The market demands XML, in and out, and support of tagged PDF and EPUB. Then, he took a look at the development of speed and memory and how we can benefit today, such as saving time struggling with hardware constraints. What if we had this 20 years ago? What would be our position today? We compete with WYSIWYG and real-time rendering, so we should explore today's hardware benefits with more effort — though constraints can lead to better solutions.

He finished with a look at perspectives. Will TEX become a niche or go mainstream? Will it be a backend? Or more used by individuals? Requirement for quality doesn't grow; other apps can do well too. Can we focus on control, and on cultural aspects? The future is not clear.

A short break followed, with some ice cream in the hotel cafe. Joseph Wright then gave an update on the status of the UK TEX FAQ. The original Cambridge server is not available any more. So, the FAQ has moved to another server, maintained by me. The original and established domain name (`tex.ac.uk`) has been preserved. Now, while I like to help in continuing, improving accessibility and web design, the UK TUG team will continue maintaining it. Joseph Wright asked for people to help improve the FAQ content. This work load can be shared, so anybody could focus on a specific part. The FAQ sources are on `github` for further development.

Joachim Schrod then described to us the services provided by CTAN. An essential part is providing LATEX and other packages to chosen servers, which in turn sync them to about 200 mirrors. That's for us TEX users, who can then update our packages. This is a principal task of all the CTAN-related servers, noticeable by the many terabytes moving. Other services, such as manually browsing package directories, are less used, as this is done more by developers than end users. Archiving and mirroring involves challenges such as observing mirror server status and checking if they are up to date.

The heart of CTAN is the TEX catalogue. Maintenance of packages' metadata is a laborious but fundamental part of the archive. Besides incoming mirroring, there are services such as the web server, including upload management, and mailing lists. With a look at the load on the CTAN servers, Joachim Schrod confirmed Hans Hagen's words that TEX may be a niche — but it's a large one.

Finally, Barbara Beeton and Volker RW Schaa gave words in memoriam of Pierre MacKay, Richard Southall and Hermann Zapf, who have passed away.

In the evening, there was a dinner in the Dreiklang restaurant. The complete LATEX3 team, Henri Menke and I decided to go to the Ratskeller. We talked about current LATEX3 development until late into the night.

TUG 2015 — day 2

Pavneet Arora started the first session with a talk about FLUSS, a flow leak monitoring system. I was curious, how this should be related to TEX. The working title, FLUSS, is an acronym. It stands for "Flow leaks unearthed ss" where ss means 2x sigma. The latter refers to double sigma testing.

His talk had little to do with typesetting. But it has to do with TEX. Pavneet considers TEX to be a part of the core stack of embedded systems. He uses TEX as a sophisticated documentation backend, and for reporting, so in this case not for publishing. So to say, he used the "TEX of Things" to detect water leaks. Why is this important? As with fire, water damage can be limited by catching the problem early on; you don't have fire detectors at home, but smoke detectors, to get warned at an early stage. He focused on the water supply instead of all possible breaks and leaks along the whole supply way. His application suite is monitoring the flow at the source side, e.g., near the water meter. It learns water consumption patterns over time, and results in ConTEXt-generated reports. They allow the triggering of alarms thanks to pattern recognition. The hardware is an embedded system based on a Raspberry Pi. There's a bunch of tools to install — TEX was the easiest part, a great sign of its maturity and its reliable packaging. We saw ConTEXt-generated diagrams and how to detect a water leak there. This topic is highly important to insurance companies, connected to much money. Thus, Pavneet showed an interesting and unexpected use of TEX in that industry.

In the next talk, Tom Hejda spoke about preparing LATEX document classes and templates for the Czech Technical University in Prague (CTU). He spoke about differences in creating classes for journal articles compared to university theses. There, he considered the user's point of view, stated some basic facts and gave examples. He started with typical usage. The procedures are different:

- Journal article: author typesets, it's reviewed, there's a final author version, it's copyedited and typeset.
- Thesis: student typesets, supervisor comments, the final version is submitted by the student.

A journal has its style, decides which packages can be used, etc.; the journal has full control of output. In contrast, with a thesis, the university has style restrictions, but the students mostly decide how to actually typeset the thesis. Journal articles and theses also differ in sectioning depths, used packages, and in the variety of topics, which is comparatively narrow in most journals.

Tom compared these examples and discussed differences in their approach:

- `actapoly`, a class for journal articles in the *Acta Polytechnica*, written in a mixture of TEX and LATEX 2ε.
- `ctuthesis`, written using LATEX3 as much as possible, with a rich key=value interface.

Boris Veytsman followed, presenting a new multibibliography package. There's actually a package with this name. He reworked it, and his new package is called `nmbib`.

Generally, a bibliography is not merely a technical list. It describes the state of the field. So, not only an alphabetical listing, but also a chronological list shows the development and progress in the field. With `nmbib`, you can have ordering by name, by appearance, and chronological, all in the same document. Each cite command produces entries for all lists. With the old `multibibliography` package, there were some limitations, such as support for just fixed BIBTEX styles. Perl was required. With the new `nmbib`, you still get a look and feel similar to `multibibliography`: you get three lists with `hyperref` links. But now `nmbib` has compatibility with (and in fact loads) the `natbib` package and supports its commands. Any `natbib` style may be used for alphabetical or sequential bibliography lists. You don't need Perl any more. Instead of using a Perl script, BIBTEX is simply run three times for three orderings. `nmbib` is much more flexible compared to `multibibliography`, since all `natbib` customizations can be used, and citation styles can be customized. The new and more flexible `nmbib` package has also been developed with ebook usage in mind.

Leila Akhmadeeva joined Boris for a presentation about trilingual templates for an educational institute in Bashkortostan, Russia. This is a special challenge because Bashkir Cyrillic is different from Russian Cyrillic. A formal document is already a challenge for a style designer, and a consistent multilingual style is even more so. TEX is a good tool for such tasks. They chose Paratype for consistent fonts.

Paul Gessler followed with a talk about printing Git commit history graphs. Git is a popular version control system. Based on the `gitinfo2` package, Paul wrote an experimental package called `gittree`, which generates such graphs for use in LATEX, on the basis of Ti*k*Z, and provides a convenient interface. He showed use and creative abuse, such as with a `github` project MetroGit where each commit is a metro station, and a branch is a metro line, a merge is a connection between lines; together, it produces a map of the metro stations of Paris. Paul's code will be on `github` by the end of summer, and he expects to put it on CTAN in early 2016.

Steve Peter then did his final task as a president: introducing the new president, Kaveh Bazargan.

Kaveh said that it's an honor to hopefully contribute in this new way to our great community. He has worked with TEX since 1983, and his first TUG meeting was 1986 in Strasbourg. He said that TEX deserves to get far more visibility. So, he hopes we keep old friends but get more youngsters to join in. We can show there are many things TEX can do which can still hardly be done using other technology, although applications have essentially caught up in many respects. He thanked everyone for voting in the election, and announced that he will talk afterwards with the people who voted differently ... with a smile. The new TUG board then gathered before the audience. (The following report on the annual meeting includes contributions from Boris Veytsman.)

A first suggestion, quickly approved, was the founding of an accessibility working group. Klaus Höppner said we could join forces with institutes working on tools for blind people. Many people are working on things such as tagged PDF; teams should be brought together.

It was said and agreed that we should demonstrate more vigorously how TEX is used in academic work and industry. But where to show it? At TUG meetings, we are talking among ourselves. The TUG web site is visited mostly by TEX users too, probably not by not-yet-users. There is a TEX showcase there.

My silent thought was that I could adapt and expand the TEXample gallery, a tagged and categorized gallery built by Kjell Magne Fauske. It's currently focused on Ti*k*Z, but could be extended to TEX in general. There are sophisticated back-end scripts for automated workflows including compiling, adding to file shares and database, tagging, and generating output in PNG and JPG via Ghostscript for gallery view and thumbnail preview.

Frank Mittelbach said that we should support the entry level at universities. Some opinions went further: we should promote very early use of TEX, such as in schools. The potential TEX entry point today is often when people start writing their thesis.

But by that time, they have already used Word and such for 10 years. TEX comes late here. Few people feel the need to switch from Word after using it for six or ten years.

Supporting this thesis, Rajagopal CV told about his experience in teaching in India, at the B.Sc. and M.Sc. levels at the University of Kerala. (We hope a detailed article will be forthcoming.)

Regarding publishers and TEX: few publishers use TEX — they were in the room, notably River Valley Technologies and VTEX. Many publishers deny TEX because there's still an old bad reputation. More and more other applications catch up. 95 percent of files coming from authors are in Word, so the industry has developed clever things around Word, expensive tools to work with Word, such as taking out references for processing. The industry standard is conversion from Word, period. TEX is in the minority. Though, it's a big industry to tackle, if you know how.

We should promote the information that the TEX distribution is actively maintained, and will continue to be. That's an important criterion. How many people go to conferences in other fields, to tell about TEX? Not so many. Of course, other user groups are also mostly among themselves.

Boris Veytsman started a discussion in another direction. Whatever we all think, where is TEX going to be in the future? And how about TUG? In past times, there was much meaning and reason for being a user group. Without today's Internet, we promoted and helped users. But today? We may have made ourselves unneeded, because we did a good job. There's CTAN, a user doesn't need to be a TUG member any more to get all the software, online help in forums, and most of the membership benefits. What reason is left to join? Thanking and sponsoring, what else? Many members join because of sympathy. We should find a justification for the TEX Users Group to exist, as such, find a convincing reason for people to join. This was an open discussion with many people contributing. Why do we need our group? How can we tell anybody that we are relevant? Should and could we find a new identity? Why is a user group necessary?

Brought up by Matthew Skala: TEX is not the first choice even in the open source world. A user buys a new computer with Ubuntu and just clicks on "Create a new document" icon. Ten times out of ten the system will open an OpenOffice or LibreOffice clone of Word: is this inevitable?

Still, even though millions of users rely on TEX, things can easily break. As wonderful as they are, teams are small. The CTAN maintainers' group consists of four people, the LATEX team consists of five people, and so on. If a key person gets ill, everything stops. Rarely do new people pick up. It's not only about users or money — an important issue is getting users to contribute and to turn into developers. We are seriously low in developers: we need users to turn into being developers. So, user groups are essential to activate people who start contributing.

That was a serious discussion, and it's good to bring up such points; to raise questions to find answers. Nevertheless, to be sure, people here are positive and in a great mood.

Away from the TEX front, in the afternoon, there was an excursion to the Messel Pit (`https://en.wikipedia.org/wiki/Messel_pit`), a UNESCO World Heritage site because of its abundance of fossils. In the evening we met at the Herrengarten and talked until late.

TUG 2015 — day 3 — first part

The third day was opened by Kaveh Bazargan and Jagath AR. They talked about today's requirements of publishers who demand XML. Kaveh showed issues with XML. He gave examples of proper XML encoding but crazy meaning, such as embedding each letter within XML text or writing a plus-minus sign by a plus symbol with an underline tag. He reviewed the classic publishing chain, from author to publisher to peer reviewer to copy editor and finally to the typesetter with possible loops. Then he showed the cloud approach, which is not so linear but more star-like: when publishing in the cloud, the XML file is in the middle while the involved parties all work directly with that file. He showed an online editor on the River Valley Technoogies platform which allows editing, reviewing and correcting with a rich online editor. There, the file is always saved in XML and rendered into HTML or PDF on the fly. Authors are editing XML, but TEX is used in the background. Specifically, TEX is used for pagination of XML documents and producing high-quality and even enriched PDF output with different styles from the same XML base code. Jagath AR showed some examples of enriched PDF, such as PDFs with several layers for screen color mode, black and white, and CMYK coloring, all in the same PDF file.

Joachim Schrod then gave an experience report about TEX in a commercial setting. The purpose is producing all written communication for an online bank. This means usually small documents, but counted in the millions, with severe legal requirements. Such document types are letters with standardized or individual content, PIN/TAN letters, account statements, credit card statements, share

notes, and so on. Some may contain forms, some contain PDF attachments of third parties. The client actually types LaTeX, but within templates: only simple LaTeX is used by the client, no math, and there are only three special characters: backslash and braces. You can imagine that common TeX symbols can have a very different meaning in this context: think of $ and % in a bank. The client uses a web application basing on a reduced tinyMCE editor. Usage has to be simple, with low latency, and it needs to be restricted for production. There are just a few special environments, tailored to the corporate identity style to ease use. The output is generated to different channels, such as to a PDF file (with letterhead), printed letters (without letterhead, as it's already pre-printed on the paper), and it needs to be archived.

Besides manually written letters, there are jobs for automated mass production, such as producing account statements each month. The standard processing steps are: generate, format, output, and archive. The engine is plugin-based, using document parameters and templates. Different representations need to be produced, such as draft, online with letterhead, on paper without, as mentioned above. Calibration to in-house printers may be needed, and additionally inserted empty pages when sending to a printshop. Folding machine control has to be implemented, different archiving needs to be supported. Everything has to be done after formatting, since there is a legal requirement to be able to reproduce the archived file in any style on any output channel even after many years. So you need a storage strategy.

In this environment, they use the classic DVI format with `\special` commands. There are different DVI drivers for each purpose. DVI is better suited since the documents are much smaller documents than PDF files, and storage costs money. The interactive preview has to be fast, small jobs have to be processed quickly in high amounts. So they use a TeX `.fmt` file with preprocessed macros. There is not even a document class selection, it's preloaded, no standard packages are used as they are preloaded with the format as well. The packages are very short, with essentially no code, just selection. Compiling a document goes down from 1.5 s to 0.06 s per document, for example, which is a factor of about 25. This is a big thing in mass production. A sample requirement is to generate and format 400,000 documents in a determined time.

To make production even more efficient, tabular material is typeset using `\vbox` and `\hbox` instead of LaTeX tabular environments. So, TeX is very fast in this regard. Jobs can be parallelized. Codes are actually piped into a TeX process. Instead of running TeX on each small file, large container files are generated with, say, 50,000 documents for a single TeX run. The DVI file then gets split in the postprocessing. Every file, all graphics used, and all fonts used get a timestamp for storing and reproducing.

The whole process has to be robust and reliable, fast, and it should use low resources such as memory and storage. The whole setting shows that traditional TeX is still useful today, even outside of academia and publishing.

The next talk by S.K. Venkatesan presented TeX as a 3-stage rocket. The stages are:

- breaking paragraphs into lines;
- making a single long scroll page;
- cutting the scroll into pages with a cookie-cutter algorithm.

With infinitely long pages, footnotes are placed after the text paragraph.

He compared creating paragraphs with CSS in HTML for browsers and as generated by TeX, and spoke about coexistence of TeX and HTML.

After a break, Joseph Wright followed with a talk about the `\parshape` primitive command, with a live demo instead of slides. The LaTeX3 team developed a new interface to `\parshape` based on three different concepts: margins, measure, and cutouts. He demonstrated setting margins to absolute values, and to values relative to the previous paragraph. He showed indenting lines differently within a paragraph, shaping paragraphs and producing cutouts with the new interface. An open challenge is that it's still line-based, but not based on heights of objects, or lengths.

Julien Cretel gave the next talk. It was about functional data structures in TeX. At first, he explained what Haskell is: a purely functional language. He gave an example of quicksort, and said he wanted to do algorithmic things within TeX. One could delegate this to an external program, but we often like to use TeX even if it may not be theoretically the best choice. Many of us like to solve things in TeX, instead of calling Matlab or such. At least it's a pleasant intellectual pursuit. Thus, Julien wants to implement semantics like this in TeX:

```
data Tree a = Empty | Node (Tree a) a (Tree a)
```

He asked the audience for feedback. For example, if TeX should be chosen for the implementation, or it should be done with LaTeX3. He plans to focus on a subset, wants to write algorithms in Haskell and then translate to TeX or LaTeX code.

So it was more an open discussion than a presentation. Maybe we can see an implementation next

year. Some comments from the audience.

- It's easier to implement Haskell in TEX than to implement TEX in Haskell.
- Why implement it in TEX, if you have Haskell already? As above, for the challenge.
- Arthur Reutenauer suggested working with the TEX tries used for implementing hyphenation.
- TEX is Turing complete ... we know, but this doesn't help in the *how*.
- And LuaTEX? Lua is imperative, not functional.

TUG 2015 — day 3 — second part

Hans Hagen gave the next presentation. The main points were

- How far can you go with TEX?
- Do you really want to go that far?

Hans showed fascinating examples, such as TEX-rendered text fed into MetaPost for postprocessing and then re-rendered by TEX for justification.

Another example was about "profiling" lines. For instance, there are two columns, and everything has to be on the grid. TEX has paragraphs, but not a concept of a line. It's pasting hboxes together, with heights and depths. TEX doesn't natively have columns, but you can implement them. He showed an example of boxed columns, all on the grid, including such things as inline fractions. By checking the actual content, lines could stay closer in the grid when heights and depths of elements did not collide. He implemented a profiling mechanism. In the end, he did not use it ... except for this talk.

A second item: many of us know the TEX command `\ignorespaces`. ConTEXt now has commands `\removeunwantedspaces`, `\removepunctuation` and others. Content can be marked as punctuation, or tagged in any way. So, you can remove such marked content, after typesetting.

Finally he demonstrated some peculiarities of an ASCIIMATH implementation, in which, for example, writing `o` twice becomes an infinity sign, with all challenges.

Boris Veytsman continued with a talk about controlling access to information with TEX. This is not only about security, but also simply hiding unneeded information. E.g., technical people may not need to see financial information, and conversely. So a document may contain both, but shows just the relevant part to each kind of reader. Output-level access control may be sufficient.

Meanwhile, regarding security, documents may contain an open part and a part with classified information. In this case, input-level access control is needed. Existing input control in LATEX is via `\includeonly` combined with `\include`. There are disadvantages or restrictions; for example, every such part starts a new page. With a lot of parts or involved reader classes it can quickly get complicated. Separate files may be confusing. A classic approach would be:

```
\newif\ifclassified
\ifclassified\input{classified.tex}\fi
```

Another solution is provided by the `comment` package, which provides environments for information with different audiences.

For output-level control, Boris has written the `multiaudience` package. The `beamer` class offers a similar concept — presentation and a handout mode, so also visibility control. But `multiaudience` has been developed for supporting any number of such modes, a.k.a. visibility classes. This is not for security, but for hiding boring or non-relevant parts. (He later learned about the `tagging` package, which provides similar functionality.)

Regarding security, there must be source level control. Boris showed the new tool `srcredact`, which is a Perl script with an input syntax inspired by docstrip. There are two modes, one to extract text for a partial version and the other to incorporate changes from a partial version. So there's two-way communication.

Finally, Enrico Gregorio showed examples of good and bad TEX code. He talked about the spurious space syndrome, which has bitten all TEX programmers at some time. Or even worse, the "missing required space" syndrome. Missing protection of line endings is classic. TEX friends had fun visually parsing code looking for spaces.

He talked about LATEX3 and showed various `expl3` examples. He strongly recommended `expl3`: even if it adds a thousand lines to load, it's worth it — later it will be part of a format anyway. It does have some disadvantages, e.g., code is much more verbose, and still requires understanding expansion. He thanked the LATEX3 team for their great work on `expl3`. I only can join the thanks.

At the end, we had a question and answer session. One of the most important questions: where and when will be the next TUG meeting? It will be in Toronto, from July 25–27, with optional excursions before and after.

Again, thanks to TUG and the sponsors DANTE e.V. and River Valley Technologies. And especially to Klaus Höppner, who did a great job as organizer!

◇ Stefan Kottwitz
stefan (at) texblog dot net
`http://www.latex-community.org`

In memoriam

Barbara Beeton

Over the past year, the TEX community has lost several individuals without whose contributions our landscape would be very different.

The first of these is one of TUG's past presidents, Pierre MacKay. An early adopter of TEX, Pierre's first goal was to create an environment for creation of documents in the Arabic script. As president, Pierre instituted regular reports to members in *TUGboat* [1], a practice continued by succeeding presidents to this day.

Two type designers whose work played a significant part in making TEX the quality tool it is today passed away this spring: Hermann Zapf and Richard Southall.

Hermann Zapf, who has been called the *fontifex maximus* of the world of typographic and calligraphic letters, needs no introduction to anyone in the least familiar with type design. His direct contribution to TEX consisted in designing the Euler family of fonts and, as a present for Don Knuth's 70th birthday, "reshaping" those fonts to take advantage of improvements in font technology. He is rightly listed among the TUG "notables" as "Wizard of Fonts". Less directly, his *hz* system of paragraph justification made its mark, through its TEX-based implementation by Hàn Thế Thành, whose Ph.D. dissertation launched pdfTEX. Finally, owing to collaboration on the font tuneup, he was named an honorary member of DANTE, the TEX group for German speakers. His kind cooperation with me in preparation of the English version of his DANTE presentation on that occasion [2] taught me much, about both the niceties of typography and his generous spirit; sadly, I never had the opportunity to meet him. He will be greatly missed.

Richard Southall was less well known, but he too made his mark on TEX. The sans serif fonts of the Computer Modern family owe much to his touch. Richard attended the 1984 TUG meeting, presenting a session on "First principles of typographic design for document production" [3], explaining the merits of a more austere approach to typography than is now (or was even then) the norm. Richard's typographic specialty was fonts for use in directories and other situations where recognition and absence of ambiguity are paramount.

Another individual important to the TEX community passed away in the summer of 2014: Thomas Koch, past president of DANTE, the users group for German speakers, from 1999–2001. (A memoriam for Thomas will be included in the next issue of *TUGboat*.)

Brief comments were shared at the meeting in memory of these four individuals, by Volker Schaa (Hermann Zapf), Joachim Schrod (Thomas Koch), and Barbara Beeton (Pierre MacKay and Richard Southall).

While I was writing this introduction, I learned of another passing: Kees van der Laan, one of the founders and first chair of NTG (the Dutch TEX group), succumbed to a long illness on August 24. Kees contributed freely to *TUGboat* [4], and the publications of other TEX groups on a wide range of topics, and was a frequent and welcome participant at BachoTEX gatherings.

Some remembrances in honor of these departed colleagues follow this introduction.

References

[1] Pierre MacKay, "From the President", a series of reports in *TUGboat*, linked from `http://tug.org/TUGboat/Contents/listauthor.html#MacKay,Pierre`

[2] Hermann Zapf, " My collaboration with Don Knuth and my font design work", *TUGboat* **22**:1-2, pages 26–30. `http://tug.org/TUGboat/tb22-1-2/tb70zapf.pdf`

[3] Richard Southall, "First principles of typographic design for document production", *TUGboat* **5**:2, pages 79-90. `http://tug.org/TUGboat/tb05-1/tb10south.pdf` Corrigenda: *TUGboat* **6**:1, page 6. `http://tug.org/TUGboat/tb06-1/tb11gendel.pdf`

[4] Kees van der Laan, contributions to *TUGboat*, linked from `http://tug.org/TUGboat/Contents/listauthor.html#Laan,Keesvander`

Photo courtesy of Diana Wright

Pierre MacKay, 1933–2015

Barbara Beeton

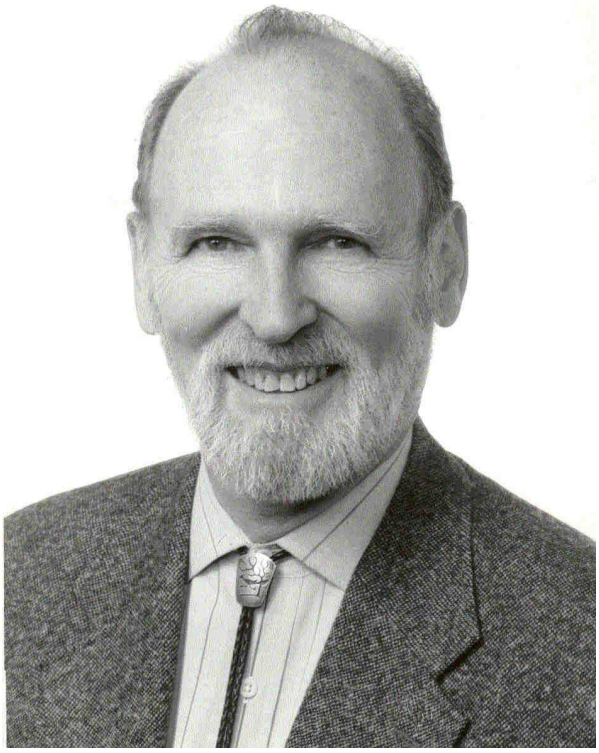
Pierre was an early adopter of TEX, and a very early member of TUG; his name first appears in a membership list in June 1981. In November 1982, he and Rick Furuta at the University of Washington assumed the duties of Unix site coordinators, a task that Pierre continued to perform until 1992.

An enthusiastic supporter of TUG, Pierre was elected president and served in that capacity during 1983–1985. He remained on the board until 1991, and was always a source of good advice and good cheer.

Pierre's primary interest was in the study of classical Greek and Arabic literature. His early studies were largely based in Greek, but needing a second classical language, he began his study of Arabic. Later, he taught himself Ottoman Turkish. Some of his first publications concerned medieval Arabic literature. In 1966, he accepted a job at the University of Washington, because their classics program included a Near East Studies program that allowed him to pursue all his main research interests while teaching the same subjects.

By the end of the 1960s, he became interested in the potential use of computers and computer languages for non-Western scripts, and was an early adopter of computerized text processing and typesetting. Working with a colleague in Ottoman studies who was an amateur Arabic-script calligrapher, he developed the first digital typesetting font in Arabic. In 1980, in addition to his positions in the departments of Classics, Near Eastern Languages and Literature, and Comparative Literature, he became an adjunct member of the Department of Computer Science. (A busy man!)

When TEX and Metafont arrived on the scene, Pierre climbed onto the bandwagon, and immediately started to work on adaptations for Arabic script. In a brief 1983 *TUGboat* announcement [1], he noted that "[for] this we must extend and modify the basic program, but we are making every effort to ensure that our Arabic Script version of TEX will be an enhancement, and will leave all the basic features of TEX82 intact." Pierre made good on his intention. In 1987, he and Don Knuth together wrote an article for *TUGboat* announcing "Mixing right-to-left texts with left-to-right texts" [2]. Lacking actual R-to-L fonts, the effect was simulated using mirrored latin Computer Modern with examples inspired by "Through the Looking Glass" (see the excerpt below). The standard conventions for interleaving and line-breaking were properly observed.



Also in 1987, the annual TUG conference was held in Seattle at the University of Washington, with Pierre as the local host. The topic that year was TEX for the Humanities, a subject dear to him.

With Pierre's urging, the university obtained various TEX-capable output devices, including an Alphatype CRS, the same machine used to produce the final camera copy of Don Knuth's *Computers & Typesetting* series. Since one of these machines was also in use at AMS, Pierre was a ready sounding board when the hardware proved recalcitrant.

Another project that he assisted, although more with moral support than active participation, was the creation of the "Washington cyrillic" font, `wncy`, through his "matchmaking" between a staff member at the UW Humanities and Arts Computing Center and the AMS, rebuilding and extending, for inclusion in the `amsfonts` collection, a proto-cyrillic font based on Computer Modern originally created at the AMS [3].

After his retirement from the University of Washington in 1990, Pierre separated himself from the "formal" TEX community in order to devote his energies to the scholarship he felt still needed doing. He could still be detected from time to time on `texhax`, sharing his copious knowledge with others who needed help. On June 14, 2015, while working on the New York Times crossword puzzle, Pierre passed away so quietly that the pencil remained in his hand.

7. Multi-level mixing. The problems of mixed R- and L-typesetting go deeper than this, because there might be an L-text inside an R-text inside an L-text. For example, we might want to typeset a paragraph whose TEX source file looks like this:

```
\R{Alice} said, \R{``You think English is \L{`English written backwards'};
but to me, \L{English} is English written backwards.  I'm sure \L{Knuth}
and \L{MacKay} will both agree with me.''}  And she was right.
```

An intelligent bidirectional reader will want this to be typeset as if it were an R-document inside an L-document. In other words, the eyes of such a reader will naturally scan some of the lines beginning at the left, and some of them beginning at the right. Here are examples of the desired output, set with two different line widths:

Alice said, "You think English is
'English written backwards'; but to me,
English is English written backwards.
I'm sure MacKay and Knuth will both
agree with me." And she was right.

Alice said, 'En-
glish written backwards'; but to me,
English is English written backwards.
I'm sure MacKay and Knuth will both
agree with me." And she was right.

(Look closely.)

Excerpt from "Mixing right-to-left texts with left-to-right texts" [2], p.17.

I treasured Pierre's friendship, and miss him greatly.

Photos and information on Pierre's non-TUG life have been supplied by Diana Gilliland Wright, who shared the last years of his life. We are greatly indebted to her for her kindness, and thank her warmly.

References

[1] Pierre MacKay, "TEX for Arabic Script", *TUGboat* **4**:2, page 72.

[2] Donald E. Knuth and Pierre MacKay, "Mixing right-to-left texts with left-to-right texts", *TUGboat* **8**:1, pages 14–25.

[3] Barbara Beeton, "Mathematical symbols and cyrillic fonts ready for distribution", *TUGboat* **6**:3, pages 126–128.

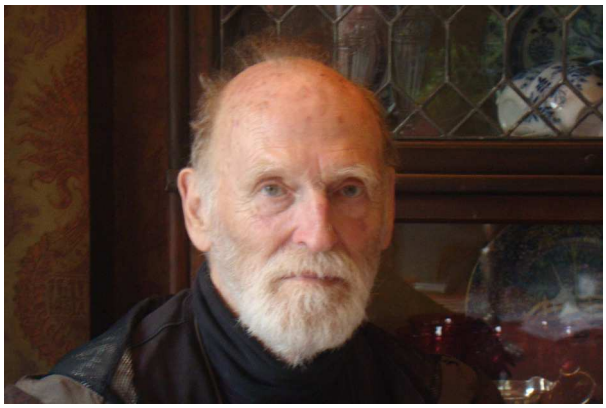
Some links to web pages telling Pierre's story:

A brief TUG interview:
`http://tug.org/interviews/mackay-p.html`

A memorial by Diana Wright:
`http://surprisedbytime.blogspot.ca/2015/06/pierre-antony-mackay.html`

A remembrance by an archaeologist familiar with Pierre's interest in Venetian and Ottoman Greece:
`https://mediterraneanworld.wordpress.com/2015/06/15/pierre-mackay/`

An account by Pierre concerning his identification of a manuscript that proved to be the missing original of a renowned work by an 18th century Ottoman author:
`http://surprisedbytime.blogspot.com/2011/06/evliyas-manuscript.html`



Dedication to Hermann Zapf, 1918–2015

Donald Knuth

The METAFONTbook is dedicated

> To Hermann Zapf:
> Whose strokes are the best

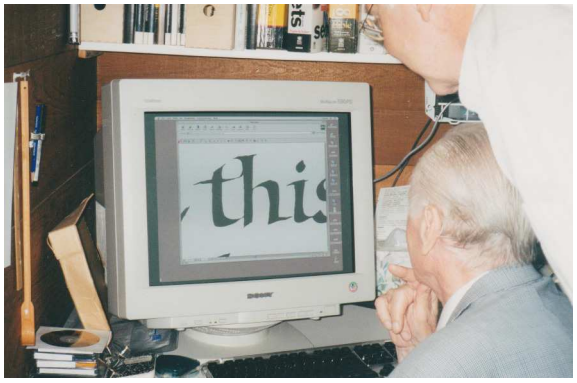
and this has an (intentional) double meaning: Not only does it say that he drew the most beautiful shapes. It secretly implies that he also gave the most valuable feedback! Because, in English, we get a "stroke" from another person when that person pays us a compliment. Hermann had very strict standards, and he never praised anything that he didn't really like. Therefore I could always count on hearing his true opinions, good or bad; this was indispensable during my years as an apprentice learning about type. I couldn't trust strokes from other colleagues, but Hermann was the best teacher because he didn't hold back criticism.

He and I collaborated closely on three projects during the 1980s: Computer Modern, Euler, and the book *3:16 Bible Texts Illuminated*.

Subsequently we corresponded rather often, and visited each other's homes several times. I could count on receiving both a Christmas card and a birthday card from him in most years! In his most recent letter (January 2015) he did say that his health was not so good.

The photos at right were taken when Hermann and Gudrun visited my house on 10 September 2001.

Hermann Zapf, Donald Knuth, and Gudrun Zapf von Hesse.



HZ and DEK at work on Don's computer.

Photos by Susie Taylor of the San Francisco Public Library.

References

[1] Hans Hagen, Taco Hoekwater, and Volker R. W. Schaa. Reshaping Euler: A collaboration with Hermann Zapf. *TUGboat*, 29(2):283–287, 2008. `http://tug.org/TUGboat/tb29-2/tb92hagen-euler.pdf`.

[2] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[3] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[4] Donald E. Knuth. *3:16 Bible Texts Illuminated*. A-R Editions, Inc., 801 Deming Way, Madison, WI 53717-1903, USA, 1991.

◇ Donald Knuth
Stanford, California

Editor's note: Requested by the editors of *Eutypon* (`eutypon.gr`), and to appear in their next issue. Kindly allowed to be "pre"-printed here by the editors and author.

Farewell Hermann Zapf

Hàn Thế Thành

I met Zapf for the first time at a TEX conference. That time I was too shy to come close and make any conversation. I was impressed how a kind and bright gentleman he was, despite his fame and age. Before that, I only knew the name Hermann Zapf from his typefaces which I used and admired.

Then, through the work on HZ extensions for pdfTEX I learned more about Hermann Zapf and also had the opportunities to contact him. During testing of the HZ extensions I sent the sample output to him, asking for his opinion. Zapf answered with a letter in his unique style, with beautiful typography, very kind and contented words. I was encouraged by his letter very much; before that I was having doubts if my work makes any sense. Later the faculty asked Zapf whether he would be willing to review my thesis.[1] Zapf accepted. I could only be thankful.

During another TEX conference in Darmstadt I met Hermann Zapf again, and was lucky enough to have a dinner with him, together with Hans Hagen and Volker Schaa. I forgot that I was having dinner with a legendary person. I was simply enjoying the time being there, for Hermann Zapf was such a pleasant person to be with. It was like being with a wise and kind uncle.

When I read the news that Hermann Zapf passed away, I had an empty feeling. I said to myself, it's an unavoidable ending of a great story. Then slowly after that I realized what a loss it was, and how sad I am.

Farewell Hermann Zapf, we will miss you until we meet again.

[1] Hàn Thế Thành, "Micro-typographic extensions to the TEX typesetting system", dissertation, Faculty of Informatics, Masaryk University Brno. *TUGboat* 21:4, 2000. http://tug.org/TUGboat/tb21-4/tb69thanh.pdf

Remembering Hermann Zapf

Kris Holmes *

Hermann Zapf is admired as the greatest type designer of his era. Vast reams have been written about his work and many more will be written. Now he is gone and I am sad.

I first met Hermann Zapf in 1979, when I studied calligraphy and type design with him in summer courses at RIT. Our acquaintance continued and became friendship over the next 36 years. In remembering him now, I want to tell the one story that means the most to me.

The German firm of Dr. Ing. Rudolf Hell, noted for their ultra-high-quality digital scanning, color, and image engraving systems, also invented the first digital typesetting machine, the Digiset. Dr. Hell made several generations of high-quality Digiset machines for the European market from the 1960s to the 1990s.

In 1981 the firm wanted to introduce a new, small Digiset typesetter into the US with original typefaces to showcase its high resolution and quality. My studio partner, Chuck Bigelow, had been retained as their American consultant. Their European head type adviser, Max Caflisch, who had worked with and learned from Jan Tschichold, was impressed by my brush-written Roman caps, which I had learned from Ed Catich and Bob Palladino. I was asked to submit proposals for new typeface designs to be reviewed by Dr. Hell's type review board for possible development and inclusion in Dr. Hell's type library. The review board included Hermann Zapf, who had already designed several original faces for the firm, including Marconi, Edison, and Aurelia, among the very first original typefaces for digital typography.

And so it was that in the fall of 1981 we flew to Basel, Switzerland, for the selection of new type designs. On the morning of the type review board, I nervously entered the room where Hermann Zapf and the other European designers sat. I handed out the type specimens I had laboriously drawn, photocopied, assembled, and prepared by hand. One of those proposals was for a new connecting script design that featured a unique inner hairline and flowing swashes like sarabands in the capitals. I provisionally named it 'Isadora' after the pathbreaking American modern dancer, Isadora Duncan whose graceful vision I admired. Isadora the typeface was a new, original, and daring design. I was afraid it might be too daring.

Each of the European designers scrutinized my samples in silence. One of them, not Hermann, had previously and privately expressed to me his doubts that a "girl" could design an original typeface, and indeed that morning I noticed a sour look on the doubter's face as he pondered my much labored-over Isadora. Yet, before the doubter could say a word, Hermann Zapf spoke instead, saying: "Yes, this is highest quality, without question. We will take it." Hermann put down my sample and nodded to me approvingly. The other designers looked at Hermann, looked at me, then looked at Hermann again and nodded their heads in obedient agreement. Isadora was accepted unanimously.

* Reprinted by permission from http://bigelowandholmes.typepad.com/bigelow-holmes/2015/06/remembering-hermann-zapf.html.

And so it was with the approving words of the great and persuasive Hermann Zapf, followed by those obedient nods from the others, including the doubter, that my life as an original typeface designer began in earnest. Dr. Hell eventually merged with Linotype, and the typeface Isadora was acquired by ITC. It is still a popular typeface today, more than 30 years later. Hermann was right.

A few weeks after that fateful meeting, a package arrived in the mail for me. It was a copy of *Feder Und Stichel* ('Pen and Graver'), Zapf's superb calligraphy engraved by August Rosenberger and printed at the Stempel type foundry. Hermann Zapf had inscribed it to me. That rare book is still my most prized possession. It was his generous way of honoring my beginning as an original designer. He certainly knew from the beautiful type designs of his wonderful, talented wife, Gudrun, that a girl could design original typefaces. I have the fondest memories of Hermann Zapf over all the years I knew him. I sent him often happy but occasionally despairing letters over the years, which he always, always answered with kind guidance.

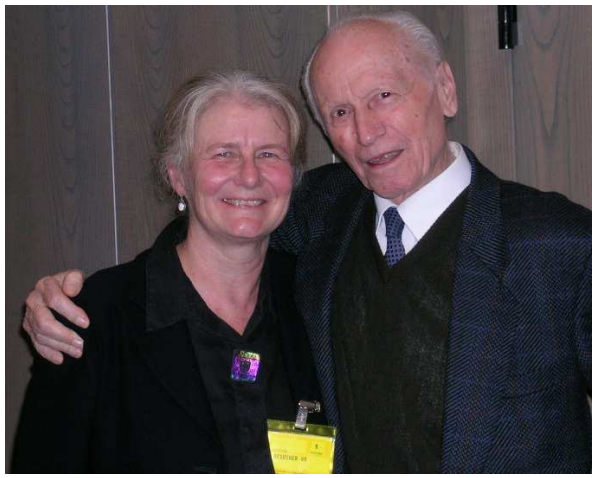
Everyone knows what a great designer Hermann Zapf was. But he was also a generous mentor, a brilliant mind, a stunning penman and a brave fighter for original design.

I loved that guy.

◇ Kris Holmes
lucidafonts.com

Hermann Zapf and his calligraphy and type design classes at RIT, summer 1979. Zapf stands in bright light, while the rest of us are in darkness. I am in the blue dress, standing above the kneeling man in sunglasses (Ned Bunnell). Behind and to my left is Jerry Kelly, talented calligrapher, book designer, and author of *About More Alphabets*, covering Zapf's most recent type designs. Behind and to my right is Chuck Bigelow, my long-time studio partner and co-designer of the Lucida fonts. On the far left, the kneeling woman is the late Dorothy Dehn, accomplished calligrapher and teacher.

I watch Zapf closely. So does Julian Waters at lower left.



With Hermann as jurors at the Linotype Arabic Type contest, 2005.

The same two of us, more light, more joking, more motion blur.

Digital typography with Hermann Zapf

Peter Karow

Hermann Zapf was involved in many of the demands for digital typefaces which came into existence from 1972 through 1997. These issues included formats, variations, interpolation, rasterizing, hinting, and grayscaling.

Within modern text composition, digital text is a special part that should proceed without manual assistance and human layout. Up to now, the milestones were these: kerning, optical scaling, paragraph composition (hz-program), and chapter composition (chapter fit).

Digital typefaces

In 1972, creative typesetting professionals produced variations manually using their photographic experience and a phototypesetting machine to make fonts with contours and shadows. Naturally, shadowing as well as contouring were my next features, which I started on 26 February 1973. This date might be regarded as the birth of "digital typefaces": new forms were generated automatically (see Figure 1).

My programs crashed quite often and generated substandard looking results. But anyhow, I got it flying further and further. That influenced my decision to call the software **Ikarus** [6] in 1974.

The first time I met Hermann Zapf was in the summer of 1975. I had just completed the basic parts of the Ikarus program which could already generate "digital typefaces" in the form of contoured, slanted and shadowed versions based on hand-digitized outlines.

Zapf was amazed to see what I could present to him (see Figures 1 and 2). He decided to include me with a presentation of Ikarus in his general presentation which he had to give at the next ATypI conference in the fall of 1975 in Warsaw.

It began a close relationship between us. This had a very decisive influence on the future development of our company URW Software & Type GmbH.

These days, everyone regards all fonts on computers as digital fonts since they are stored in digital formats such as **OpenType**. In the early seventies, we had long discussions with famous designers. They argued that pure mapping from analogue to digital is not changing the basic quality of a typeface (old properties), namely its type, appearance, effect, expression and congeniality. Therefore they asserted that the typefaces only had digital images and were therefore still analogue. "Digital" at that time was regarded as a pseudo-property.

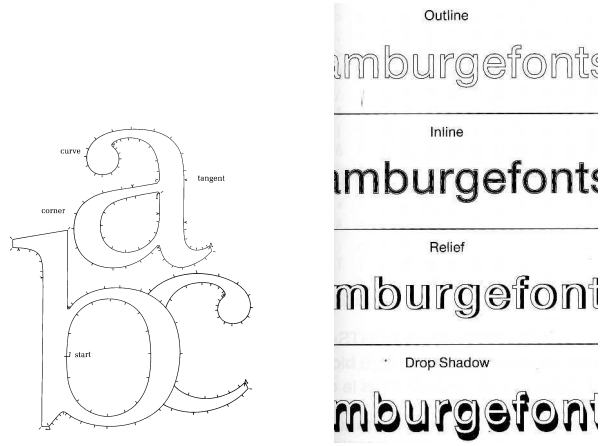


Figure 1: The first version of Ikarus-Format as started in 1972 (left). Contouring used to make Outline and Inline versions, shadowing and contouring used to make Relief and Drop Shadow (right).

Figure 2: Typical desktop in 1972 with digitizer tablet, "mouse", keyboard (below), direct VDT, alphanumeric terminal, electronic digitizer.

Never did I hear such arguments from Hermann Zapf; he was already very familiar with computers and always very eager and demanding to see the next innovation.

Today, the property "digital" is not only accepted but also embraced. It serves as an additional characteristic which doesn't interfere with the old properties and holds an extremely high significance regarding a font. It allows and creates new and important functions which did not exist before.

With the invention of the Digiset by Rudolf Hell in 1965, typefaces were digitized for the first time. No additional ideas were put into place other than using them 1:1 for typesetting on the Digiset, scaled linearly and displayed at resolutions between 1,000 and 2,400 dpi. Hermann Zapf was engaged by Hell to consult on typesetting and to design new typefaces.

Figure 3: Kerning can be regarded as a power that repulses characters the nearer they come to each other, and that attracts characters the farther they get from each other (left). Kerning can be used also to calculate character positions for overlapping and touching of text.

In 1985 during the yearly ATypI conference (this time sponsored by Hell in Kiel) Hermann Zapf discussed with me the problem that too many people were talking about digital typefaces and unfortunately not knowing what they are really all about. So, I decided to write a book with the title "Digital Formats for Typefaces" which he kindly corrected as a co-editor. Finally, we could present it in 1986 during the next ATypI conference in New York. Later I changed the title: it became "Digital Typefaces" [3].

Digital text

Being pushed by Hermann Zapf, I started automatic **kerning** [4, 7] in 1980 together with Margret Albrecht. We wanted to save money because the generation of kerning tables along with left and right side bearings took a lot of time in our typeface production. As in other cases of artificial intelligence, we had to go through several approaches throughout the years until 1987. Finally, we mixed programmed ideas and heuristic parameters gained by processing a lot of existing kerning tables manufactured by different companies. We expanded kerning to get overlapping and blending of characters in tightly composed words [9] (see Figure 3).

In hot metal printing, **optical scaling** was usual [2]. In any case one had to cut the point sizes individually, so it was a matter of knowledge, but not of money. This changed when phototypesetting came up and the possibility of linear scaling came into existence. Optical scaling didn't play a role at the beginning of DTP, however, a lot of people wrote about it [1]. Hermann Zapf urged me to "do something".

In 1991 at URW, we adopted the following approach for text fonts. First, the smaller the type size:

1) the wider the composition,
2) the thicker the strokes,
3) the broader the characters, especially the lowercase.

And second, the larger the type size in titles:

1) the more compact the composition,
2) the thinner the strokes, especially the hairlines,
3) the narrower the characters, especially the lowercase.

Simplified, one applies the rule that space and stroke width of light fonts (text fonts) are reduced or enlarged by 7% on the average if the point size is enlarged or reduced by a factor of 2 [8] (see Figure 4). For bold fonts the opposite is true.

To my knowledge, optical scaling was not employed for bold fonts in the past because they weren't (and still aren't) used very often, and if so, they were cut just for these special cases in certain point sizes as a special effort.

Paragraph composition was our next project in 1990 and it became the favorite of both of us.

We called the program **hz-engine**. Developed with and named after Hermann Zapf [5, 12] — it uses a justification per paragraph system — as described by Donald E. Knuth [10, 11], along with "kerning

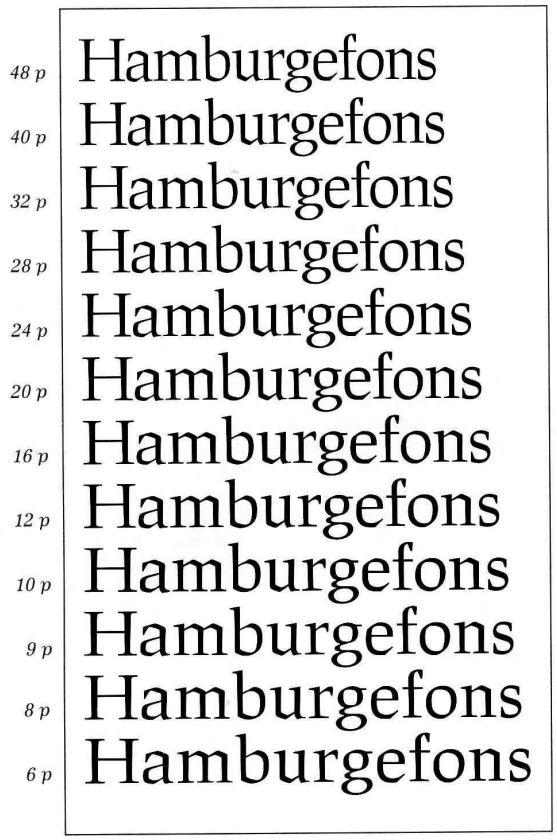


Figure 4: Different point sizes, which have been generated at the same size for the purpose of comparison.

on the fly" and expanding/condensing of characters in order to obtain margin lines for a column that are optically straight (optical margins), and achieve typeset spaces among words within lines of text that are fairly constant in order to avoid rivers and creeks.

Rivers run vertically through poorly spaced words in consecutive lines of text when the spaces between the words have the same space or a greater space than the distance between the baselines of the text. In contrast, a creek is a less severe form where the spaces between words are accidentally too wide within one line.

The basic feature of the *hz*-engine, which was programmed by Margret Albrecht, is to regard all lines of a given paragraph at once — making the **justification per paragraph**. At first, all words or syllables are distributed to the lines together in a manner where each line gets a line length nearest to its given individually parametrized width (as default there is usually column width). The following optimization is controlled by minimizing the typographical demerits, which are obtained from a function of the actual line lengths, given line lengths, given line widths and tolerances of the layout parameters.

If hyphenation is turned on, words are replaced by syllables. The *hz*-engine has to follow a lot of exceptions and to provide solutions for them, e.g. ligature substitution, consecutive hyphens and good or bad locations for hyphenation within a word. This level of text/typographic detail promotes a better fit and contributes to the reader's comfort (see Figure 5).

A comparison between the *hz*-engine and today's typical composition tools demonstrates the superiority of the *hz*-engine (see Figure 6). In 1995, the "*hz*-engine" was implemented in **InDesign** by Adobe. This has been a big step in digital text composition.

I believe that not too many people learned that much about the "digital side" of Hermann Zapf as Don Knuth and me. I learned a lot from him and thank him.

References

[1] André, A., Vatton, I.: "Dynamic Optical Scaling and Variable-Sized Characters", Electronic Publishing 7(4): 231–250, 1994

[2] Johnson, B., "Optical Scaling", Master's Thesis for RIT, Rochester, New York, 1994

[3] Karow, P., "Digital Typefaces", Springer Verlag, Heidelberg, 1994

[4] Karow, P., "Font Technology", Springer Verlag, Heidelberg, 1994

[5] Karow, P., "*hz*-program", brochure, URW Verlag, Hamburg, 1992

[6] Karow, P., "Ikarus", German brochure, URW Verlag, Hamburg, 1975

[7] Karow, P., "Kernus", brochure, URW Verlag, Hamburg, 1993

[8] Karow, P., "Optical Scaling", brochure, URW Verlag, Hamburg, 1991

[9] Karow, P., "Set theory for characters", brochure, URW Verlag, Hamburg, 1987

[10] Knuth, D. E., Plass, M. F., "Breaking paragraphs into lines", Software—Practice & Experience, 11/11), 1119–1184, Nov. 1982

[11] Knuth, D. E., "Mathematical Typography", Stanford University, Cal., February 1978

[12] Zapf, H., *Alphabet Stories — A Chronicle of Technical Developments*, Mergenthaler Edition Linotype GmbH, Bad Homburg, 2007

◇ Peter Karow
Hamburg, Germany

His Secret

Hyphenation turned off.

To the left the hz-program: 38 lines, last lines of paragraphs ok.

To the right today's software: 40 lines, short last lines, larger spaces.

What makes the Gutenberg Bible the unattainable masterpiece of the art of printing? The printing on his handpress? Can't be really, because of today's standards, the inking was not of extraordinary quality. We could order hand made rag paper also in our day. Maybe the secret of his beautiful pages is in the proportions of the columns on the paper. But this we are also able to copy. Therefore only the composition is to be considered closely.

How could Gutenberg get those even gray areas of columns without disturbing or unsightly holes between words? His secret: the master achieved this perfection by applying several characters of different width combined with many ligatures and abbreviations out of his type case. He finally created 290 characters for the composition of the 42-line Bible. An enormous time consuming job to realize his idea of good typographic lines: the justified lines of even length, compared to the flush-left lines of the works of the famous mediaeval scribes.

But with Johannes Gutenberg's unusual ligatures and abbreviations, today we can't apply this old principle for contemporary composition. Now we can get help through the versatility of modern electronic software and formats like the Multiple Masters to receive a perfect type setting in our production, to achieve Gutenberg's standards of quality: The hz-program, named after Hermann Zapf.

What makes the Gutenberg Bible the unattainable masterpiece of the art of printing? The printing on his handpress? Can't be really, because of today's standards, the inking was not of extraordinary quality. We could order hand made rag paper also in our day. Maybe the secret of his beautiful pages is in the proportions of the columns on the paper. But this we are also able to copy. Therefore only the composition is to be considered closely.

← *too short*

How could Gutenberg get those even gray areas of columns without disturbing or unsightly holes between words? His secret: the master achieved this perfection by applying several characters of different width combined with many ligatures and abbreviations out of his type case. He finally created 290 characters for the composition of the 42-line Bible. An enormous time consuming job to realize his idea of good typographic lines: the justified lines of even length, compared to the flush-left lines of the works of the famous mediaeval scribes.

← *a creek*

But with Johannes Gutenberg's unusual ligatures and abbreviations, today we can't apply this old principle for contemporary composition. Now we can get help through the versatility of modern electronic software and formats like the Multiple Masters to receive a perfect type setting in our production, to achieve Gutenberg's standards of quality: The hz-program, named after Hermann Zapf.

← *too short*

Figure 5: Comparison between hz-engine and standard composition, without hyphenation.

Figure 6: Comparison between hz-engine and standard composition, narrow columns, with and without hyphenation.

Richard Southall: 1937–2015

Jacques André and Alan Marshall

Richard Southall, typographer, teacher, scholar and well-known specialist in the field of digital typography, passed away on May 26th, 2015, at the age of 78.

In 2005, Richard published an important study entitled *Printer's type in the twentieth century* in which he outlined his own biography [21, p. xiv]:

> The author joined Crosfield Electronics Ltd in north London in the summer of 1965, to work on the specification of matrices for Lumitype-Photon photo composing machines. At the end of the 1960s he found himself in New England, building a large high-resolution camera for photo matrix manufacture at Photon Inc. In the early 1970s he was responsible for the typography of Crosfield's novel scanned-matrix photo composing machine, the Magnaset 226. At the University of Reading later in the same decade he designed a series of directly generated subtitling fonts for broadcast television. In the 1980s he worked in California, at Stanford University, Adobe Systems Inc. and the Xerox Palo Alto Research Center. Much of this work was at the disputed frontier across which computer science and traditional typography contemplated one another in an uneasy truce. In the 1990s he was closely involved with the Colorado type making project described in Chapter 9.

Richard is typically modest in his preface, just as he was in his professional activities which are impossible to sum up under one simple heading. The various fields in which he was involved include photo- and digital typesetting, film subtitling, and the theory and history of typemaking [21, 22, 23].

After graduation from Cambridge with a degree in natural sciences in 1960, Richard worked for a while in technical publishing (for Wireless World). His initial involvement with typemaking began when he took over from Matthew Carter the job of liaison on typographical questions between the British scanner and phototypesetter manufacturer Crosfield Electronics and the French type foundry Deberny & Peignot. His job was to oversee the production of photo-matrix disks by D&P for Crosfield's version of René Higonnet and Louis Moyroud's revolutionary second generation Lumitype phototypesetter which was produced under the name Photon 540. At first sight that might seem a simple task. In fact it was extremely complex because the manufacturing process not only had to work to extremely fine tolerances, it also had to take into account myriad technical details of type design, language and typographical markets and conditions of production.

Typemaking in the 1960s and 1970s also had to cope with accelerating technical change. Phototypesetting quickly went through four distinct generations from electromechanical to digital, before desktop publishing and PostScript swept (nearly) all before them in the eighties. As printers and, increasingly, graphic designers struggled to make their way through what Lawrence Wallis once called "The phototypesetting jungle", Richard never contented himself by simply applying the new technologies empirically to the job in hand: rather, his aim was always to understand the theoretical foundations which underpin the techniques and purpose of typemaking.

Over a period of about 20 years, he was type consultant to various different groups such as the American Mathematical Society, Rank Xerox for whom he worked on machine-user interfaces, British Telecom (BT), the Civil Aviation Authority and National Air Traffic Services (in the UK) on typography for information systems used by air traffic controllers, and, most recently, US West Direct (now US West Dex, see below).

One aspect of Richard's career of particular interest to readers of *TUGboat* concerns his long-term use and exploration of TEX and METAFONT.

As he was designing Computer Modern, Don Knuth invited Richard to visit Stanford. At this time he was teaching type design at the Department of typography & graphic communication (University of Reading, UK, 1974–1983) and had an intimate knowledge of the punch cutter's skills (such as the proportions of stems and the impact of size variations on letter forms and legibility). Knuth and Richard spent the entire month of April 1982, working about 16 hours a day, revising Computer Modern from A to z. While at Stanford he also took part in early TUG conferences [4] and co-taught the new version of METAFONT with Don Knuth and Chuck Bigelow. In January 1985 he gave the first METAFONT tutorial in Rennes [5, 7] and had the opportunity to work one year at the Université Louis-Pasteur in Strasbourg (with Jacques Désarménien, who had been at Stanford in 1983–84, and Dominique Foata) where he designed a font using METAFONT [9].

Throughout this period he kept in touch with researchers in the digital field of typography through conferences such as *Raster imaging and digital typography*, the *Didot project*, and *Electronic publishing*, in which he was involved as speaker/author, as a member of scientific committee, or as editor [15].

It was a great pleasure to work with him, for he was conscientious, competent and hard to please!

Richard was not primarily a type designer. His unique knowledge and skills placed him at the meeting point of type design and computer science. And though he always said "No, I'm not a computer scientist", he understood and used computers constantly. His main question was *How to design a type*; what is the appearance of a character; how can it be transferred from the designer's head to paper; how does each character relate to the others in a given font; and if it comes to that, what is a font? The kinds of questions which bring us back to the famous *What is the a-ness of an a?* Though most of his papers addressed such fundamental questions [8, 10, 11, 12, 14], Richard never published "the" answer. However, he was sure he was right in considering that the process of type design remains to be redefined.

Richard had the opportunity of putting his ideas into action when, in 1995, he was asked, not to design a typeface, but rather to design the process of designing a typeface, when he was approached by his long-time colleague Ladislas Mandel, an eminent French type designer with whom he had worked when the latter was type director at Deberny & Peignot. Ladislas, who had subsequently gone on to design many typefaces for telephone directories, asked Richard to work with him on the production of a family of new designs for the North American telephone directory publisher US West Direct. Ladislas as the type designer and Richard as the designer of the process by which the fonts would be made, successfully produced the Colorado Directory System ([17, 18] and [21, chapter 9, pp. 204–218]). METAFONT was the key to the design process, as well as to the metrics. As Richard concluded: *The power and flexibility that a fully-parametrized formatter affords to the typographer is well worth the effort involved in building it.*

One final point: if you look at his book, *Printer's type in the twentieth century* [21] which was typeset using plain TEX and Richard's own font, the text is neither justified nor ragged. Rather, it is "lightly ragged", thanks to the subtle use of hyphenation to keep the white space at the end of the line as uniformly short as possible. He was very proud of this meticulous achievement, a characteristic of his way of working.

References

[1] Richard Southall, 'Electronics in typesetting', *Wireless World*, vol. 74, no. 1389, 1968.

[2] Richard Southall, 'Character generator systems for broadcast television', *Information Design Journal*, 2(1) 1981.

[3] Richard Southall, 'The problems of forms artwork production', London: Management & Personnel Office. HM Government. 1982.

[4] Richard Southall, 'First principles of typographic design for document production', *TUGboat*, vol. 5, no. 2, 1984, 79–90. http://tug.org/TUGboat/tb05-2/tb10south.pdf

[5] Richard Southall, 'METAFONT and the problem of type design', *Typographie et informatique : support du cours INRIA, Rennes, 21–25 janvier 1985* (J. André and P. Sallio, eds.), INRIA, Rocquencourt, 1985. http://jacques-andre.fr/japublis/RS-MS-Typedesign.pdf

[6] Richard Southall, 'Designing new typefaces with METAFONT', Stanford Computer Science Department report STAN-CS-85-1074, 1985. http://i.stanford.edu/pub/cstr/reports/cs/tr/85/1074/CS-TR-85-1074.pdf

[7] Richard Southall and Jacques André, 'Experiments in teaching METAFONT', *TEX for Scientific Documentation* (D. Lucarella, ed.), Addison-Wesley, 1985, 141–153. http://jacques-andre.fr/japublis/1985-TEDX.PDF

[8] Richard Southall, 'Shape and appearance in typeface design', *Protext III* (J.J. Miller, ed.), Dublin: Boole Press, 1986, 75–86.

[9] Richard Southall, 'Designing a new typeface with METAFONT', *TEX for scientific documentation* (J. Désarménien, ed.), Lectures Notes in Computer Science 236, Springer Verlag, 1986, 161–179. https://books.google.fr/books?hl=fr&id=GaGCGEsp7aEC

[10] Richard Southall, 'Visual structure and the transmission of meaning', *EP conference* (J.C. van Vliet, ed.), Cambridge University Press, 1988, 35–45. https://books.google.fr/books?id=lfg8AAAAIAAJ

[11] Richard Southall, 'Interfaces between the designer and the document', *Structured documents* (J. André, R. Furuta, and V. Quint, eds.), Cambridge University Press, 1989, 119–131. https://books.google.fr/books?id=BrgbosTycOgC

[12] Debra Adams and Richard Southall, 'Problems of font quality assessment', *Raster Imaging and Digital Typography* (J. André and R.D. Hersch, eds.), Cambridge University Press, 1989, 213–222. https://books.google.fr/books?id=mjO9AAAAIAAJ

[13] Richard Southall, 'Character Description Techniques in Type Manufacture', *Raster Imaging and Digital Typography II* (R.A. Morris and J. André, eds.), Cambridge University Press, 1991, 16–27; https://books.google.fr/books?id=Q9KtGcpfNgUC. Republished in *Computers and typography* (Rosemary Sassoon, ed.), Intellect Books, 1993, 83–100; https://books.google.fr/books?id=Gm4QAbGKIBgC

[14] Richard Southall, 'Presentation rules and rules of composition in the formatting of complex text', *EP'92: Proceedings of Electronic Publishing '92, International conference on electronic publishing, document manipulation and typography, Lausanne, Switzerland* (C. Vanoirbeek and G. Coray, eds.), Cambridge University Press, 1992, 275–290; https://books.google.fr/books?id=YN1sLgZtHC8C. Republished in *Computers and typography* (Rosemary Sassoon, ed.), Intellect Books, 1993, 32–51; https://books.google.fr/books?id=wdYmvQD5C8IC

[15] Jacques André, Jakob Gonczarowski, and Richard Southall, *Proceedings of the Raster Imaging and Digital Typography Conference*, special issue of *EP-ODD, Electronic Publishing–Origination, Dissemination and Design*, 6(3), 1993. Table of contents, editorial and colophon in http://jacques-andre.fr/japublis/ridt94.pdf

[16] Richard Southall, 'A survey of type design techniques before 1978', *Typography papers* vol. 2, Reading, 1997, 31–59.

[17] Richard Southall, 'METAFONT in the Rockies: The Colorado Typemaking Project'. *Electronic Publishing, Artistic Imaging, and Digital Typography — 7th International Conference on Electronic Publishing, EP'98; held jointly with the 4th International Conference on Raster Imaging and Digital Typography, RIDT'98, St. Malo, France, March 30–April 3* (R.D. Hersch, J. André, and H. Brown, eds.), Lecture Notes in Computer Science Volume 1375, Springer-Verlag, 1998, 167–180; http://link.springer.com/chapter/10.1007%2FBFb0053270. Republished in *Computers and typography 2* (Rosemary Sassoon, ed.), Intellect Books, 2002; https://books.google.fr/books?id=wdYmvQD5C8IC

[18] Richard Southall, 'Prototyping telephone directory pages with TEX', *Cahiers GUTenberg*, No. 28–29, 1998, 283–294. http://cahiers.gutenberg.eu.org/cg-bin/article/CG_1998___28-29_283_0.pdf

[19] Richard Southall, Peter Enneson, Andrew Boag, and Hrant Papazian, 'Replies to Peter Burnhill', *Typography papers*, vol. 4, Reading, 2000.

[20] Richard Southall, *Ancient BBC 'subtitling' font revealed*, 2005. http://screenfont.ca/fonts/today/TKST/

[21] Richard Southall, *Printer's Type in the Twentieth Century*, New Castle and London: Oak Knoll Press and The British Library, 2005, 256 pages.

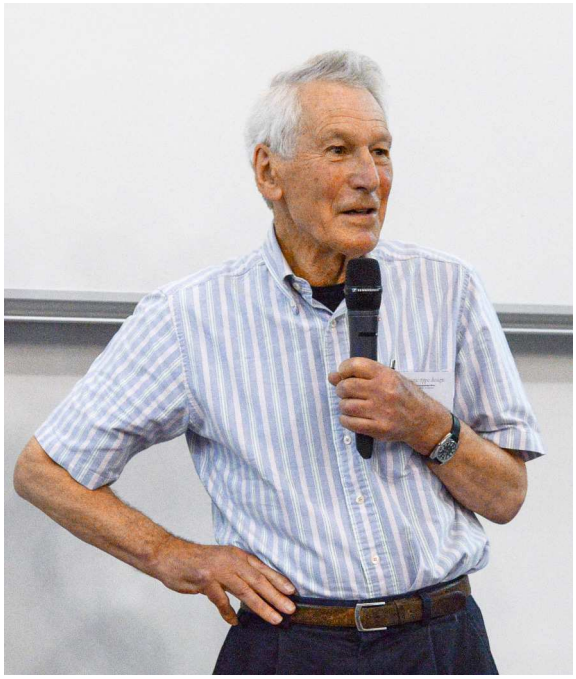
[22] Judith Slinn, Sebastian Carter, and Richard Southall, *History of the Monotype Corporation* (A. Boag & C. Burke, eds.), London, Vanbrugh Press, 2014 (esp. Ch. 2 and 3). http://vanbrughpress.com/publications-2/

[23] Richard Southall, 'The dematerialization of type', *Proceedings of the Automatic Type Design Conference* (Thomas Huot-Marchand, ed.), ANRT, Nancy, 2013. To appear.

Richard's archives

Some years ago, Richard Southall gave a large part of his professional archives to two institutions:

- The OAC (Off line Archive of California) received his papers concerning the period 1982 to 1985: the CM project, teaching material on METAFONT and digital typography and on his own TKMF and NMT (METAFONT typeface design systems): http://www.oac.cdlib.org/findaid/ark:/13030/kt487036pm/
- The Musée de l'imprimerie at Lyons: technical and promotional documents concerning the Lumitype-Photon phototypesetter (from the time when he worked for Crosfield Electronics), and a subsequent batch of documents concerning the Colorado Project including numerous samples of work in progress and printed telephone directories: http://www.imprimerie.lyon.fr/imprimerie/sections/fr/documentation/fonds/southall/



Richard Southall at ANRT conference, Nancy (France), May 2014
[Photo Michel Sabbagh]

Memories of Kees: C.G. van der Laan, 1943–2015

Erik Frambach, Jerzy Ludwichowski and Philip Taylor

Kees (really Cornelis Gerardus) van der Laan, 22-12-1943 – 24-08-2015. TEX guru, Macintosh and PostScript *aficionado*, but most of all a warm and generous man who made friends wherever he went, and who will be remembered with deep affection by all who had the pleasure and privilege of knowing him.

In 1988, Kees was one of the founders of the NTG (*Nederlandstalige TEX Gebruikersgroep*). He gathered together many other TEX enthusiasts and worked hard on building a network of people and organisations using and experimenting with TEX. Many activities were organised, such as meetings, courses and workshops. Kees persuaded several people to join him on his trips to BachoTEX and even to Russia, where lots of TEX activity was developing. Bringing together kindred spirits in order to generate even better ideas and develop them into new projects and new products was always on Kees's mind. He was a true TEX evangelist, and was made an Honorary Life Member of NTG in recognition of the quality and quantity of his contributions to the TEX world at large (his early work included TEX code for typesetting crosswords, Bridge and the Towers of Hanoi; the implementation of stacks and queues in TEX; TEX algorithms for sorting and searching; and many many other ideas as well).

In the Netherlands, he (and the NTG) actively supported a project to develop the very first plug & play CD-ROM with TEX software. This turned out to be a giant leap forward by providing a consistent TEX set-up for use both by beginners and experts alike that worked right out of the box.

In Poland, Kees was guest of honour at one of the very first GUST ("*(Polska) Grupa Użytkowników Systemu TEX*") conferences, and was formally made an Honorary Life Member of GUST at the AGM which took place during that meeting. Until then, national TEX user group meetings had tended to be just that — national — but BachoTEX set out to be a truly INTERnational user group meeting, and succeeded beyond its wildest dreams.

At that time, Kees was working on some extensions to `manmac`, and his talk at the 1994 BachoTEX meeting was entitled "Manmac BLUes". By 1995, Kees's work on extending `manmac` had grown into a complete format, "BLUe's format", as Kees called it.

Photo courtesy of Frans Goddijn

Based on `manmac` (amongst other sources), but primarily consisting of Kees's own unique work (Kees described it as "build[ing] upon `manmac`, upon functionalities provided in the *TUGboat* styles, and upon experience gained by the AMS in TEX formatting"), BLUe's format (or "BLUeTEX", as it later came to be known, probably by analogy with Blue-tack!) was publicly released at the 1995 BachoTEX meeting, and Kees gave the first of many, many talks to GUST on that subject during the conference. BLUe's format, which takes its name from Knuth's apocryphal "B. L. User" (in Britain he would be called "the man on the top deck of a Clapham Common omnibus" — i.e., the average person, as opposed to a TEX wizard), set out to make the full power of TEX accessible to the average user by encapsulating in a single format all of those elements that are essential for an advanced use of TEX but which would almost certainly be too difficult for the average TEX user to program for him/herself. Of particular interest to Kees was his wish that the user should adopt "minimal markup", and he would return to this theme time and time again.

As well as pure computer science and programming — for instance, Kees helped establish a FORTRAN interface for Algol 68, to facilitate use of existing FORTRAN libraries — Kees had a keen interest in computer graphics (a theme that was to become ever more important to him as time passed), and his 1996 papers were on "TEX and Graphics — a reappraisal of METAFONT/METAPOST" and on "Turtle graphics in TEX (a child can do it)!". During the years that followed, Kees presented papers on "TEX inside, or insights in TEX?", "A little bit of PostScript", "Tiling in PostScript and METAFONT", "Syntactic Sugar", and many, many other topics. In recent years, he combined his all-time passion for mathematics and computer graphics by creating beautiful illustrations of mathematical theorems, publishing his results in various journals and BachoTEX conference materials. He demonstrated how PostScript can

Editor's note: This note will be published simultaneously in MAPS (the NTG journal) and in the GUST *Biuletyn*.

be used to elegantly compute and display fractals of any kind. In fact, Kees showed that for any 2D, 2.5D and 3D problem that can be modelled as a mathematical equation, PostScript is a great tool to explore and solve that problem. But Kees was also an art lover. Whenever he noticed symmetries or mathematical inspiration in a piece of art, he would try to model and emulate it in PostScript. In particular, the works of Gabo, Mondriaan and Escher (minimalists all!) inspired him. And of course, this work was presented not only at BachoTEX and NTG meetings — Kees was a regular attendee at EuroTEX meetings, TUG meetings, and anywhere else where he could exchange ideas on mathematics, computer science, computer graphics and computer programming with other like-minded individuals.

Kees was always very generous in sharing whatever he had developed. He always published his work for free, and wrote more than a hundred articles in MAPS (NTG's journal, `http://ntg.nl/maps`), explaining how it all works and how it could be adapted to anyone's own needs and preferences. He was a big advocate of keeping things simple ("minimal markup" was one of his hallmarks). The TEX or Postscript code he wrote was typically very neat, concise and elegant, much in line with the programming style that Knuth (whom Kees admired greatly) used in his `manmac` macros. Sometimes his code and articles were hard to follow, but they were always worth the time and energy. There was a lot to be learned from his fresh and clever approach to solving problems.

But Kees's interests were not restricted to the technical domain — he was a great conversationalist, as well as great company, and the authors of this short tribute are just a few amongst many who have spent countless happy hours with Kees in and around his home in Garnwerd (in the early days, always accompanied by Beer-the-dog), and it was Kees who introduced many of us to the Netherlands custom of eating fresh herring dipped in onion and washed down with some good Netherlands lager. Also to *krupuk* (think "giant prawn crackers"), and to many other Netherlands delicacies as well.

Kees loved people, and made friends wherever he went. He was particularly interested in Russia, and made many very good Russian friends, some of whom would visit him in Garnwerd and stay at his home, and with one of whom, Sveta (Svetlana L Morozova) he fell in love and subsequently married. Sveta was Kees's constant companion in the later years of his life, and his greatest concern was how she would cope when he was no longer here. Sadly that time has now come, and this short series of recollections is dedicated to Sveta, in Kees's memory.

Farewell, Kees — it was an honour and a privilege to know you and to spend time with you; TEX conferences, and life in general, will never be the same again.

References

[1] CTAN. Contributor Kees van der Laan. `ctan.org/author/id/laan`.

[2] C. G. van der Laan and J. R. Luyten. Evaluation of K-talk. *TUGboat*, 9(3):271–272, November 1988. `tug.org/TUGboat/tb09-3/tb22laan.pdf`.

[3] C. G. van der Laan. Dutch TEX Users Group. *TUGboat*, 9(3):316–316, November 1988. `tug.org/TUGboat/tb09-3/tb22news.pdf`.

[4] C. G. van der Laan. Typesetting bridge via LATEX. *TUGboat*, 10(1):113–116, April 1989. `tug.org/TUGboat/tb10-1/tb23laan.pdf`.

[5] C. G. van der Laan. Announcing two reports: SGML-LATEX and Journal style guidelines. *TUGboat*, 11(1):86–86, April 1990. `tug.org/TUGboat/tb11-1/tb27laan.pdf`.

[6] Kees van der Laan. Typesetting bridge via TEX. *TUGboat*, 11(2):265–276, June 1990. `tug.org/TUGboat/tb11-2/tb28laan.pdf`.

[7] Kees van der Laan. NTG's second year. *TUGboat*, 11(3):446–447, September 1990. `tug.org/TUGboat/tb11-3/tb29reports.pdf`.

[8] Kees van der Laan. SGML (, TEX and ...). *TUGboat*, 12(1):90–104, March 1991. `tug.org/TUGboat/tb12-1/tb31laan.pdf`.

[9] Kees van der Laan. Math into BLUes. *TUGboat*, 12(3/4):485–501, November 1991. `tug.org/TUGboat/tb12-3-4/tb33laan.pdf`.

[10] Kees van der Laan. Tower of Hanoi, revisited. *TUGboat*, 13(1):91–94, April 1992. `tug.org/TUGboat/tb13-1/tb34laan.pdf`.

[11] Kees van der Laan. FIFO and LIFO sing the BLUes. *TUGboat*, 14(1):54–60, April 1993. `tug.org/TUGboat/tb14-1/tb38laan.pdf`.

[12] Kees van der Laan. Syntactic sugar. *TUGboat*, 14(3):310–318, October 1993. `tug.org/TUGboat/tb14-3/tb40laan-sugar.pdf`.

[13] Kees van der Laan. Sorting within TEX. *TUGboat*, 14(3):319–328, October 1993. `tug.org/TUGboat/tb14-3/tb40laan-sort.pdf`.

[14] Kees van der Laan. BLUe's format — the off-off alternative. *TUGboat*, 17(2):215–221, June 1996. `tug.org/TUGboat/tb17-2/tb51blue.pdf`.

[15] Kees van der Laan. Turtle graphics and TEX — a child can do it. *TUGboat*, 17(2):222–228, June 1996. `tug.org/TUGboat/tb17-2/tb51turt.pdf`.

[16] Kees van der Laan. Graphics and TEX — A reappraisal of METAFONT/MetaPost/PostScript. *TUGboat*, 17(3):269–279, September 1996. `tug.org/TUGboat/tb17-3/tb52meta.pdf`.

[17] Kees van der Laan. TEX education — a neglected approach. *TUGboat*, 30(3):5–33, 2009. `tug.org/TUGboat/tb30-3/tb96laan.pdf`.

Development of the UK TEX FAQ

Joseph Wright

English-speaking users of TEX have benefited for many years from the availability of a curated set of frequently asked questions. The origins of the current set go back to the early 1990s, when the UK TEX Users' Group (UK-TUG) published a set of questions and answers in *Baskerville*, our (now dormant) publication. The material gathered at that time was a development of regular FAQ postings to `comp.text.tex` which had been maintained by Bobby Bodenheimer.

Following the initial publication by UK-TUG, the committee took up updating the questions and answers. Whilst this task was not originally intended for a single author, for much of the past 20 years this has been undertaken by Robin Fairbairns. Indeed, the current content and structure of the FAQ is essentially Robin's work, with the contribution of UK-TUG more generally being mainly in the initial phase.

As well as providing a printable version of the FAQ on CTAN (Fairbairns, 2014) (currently over 250 pages), Robin has for many years made the information available as a searchable electronic resource: `http://www.tex.ac.uk/faq`. This online version of the FAQ is invaluable as it provides a set of stable, linkable answers to the over 400 questions in the collection.

The physical server hosting the FAQ was housed at the University of Cambridge. However, changing university practice (hosting non-official material is increasingly difficult) coupled with a change of status (Robin has retired) meant that a new home was required for the material. UK-TUG have now made arrangements to place the FAQ on a new (physical) server, maintaining the `tex.ac.uk` domain so that existing links remain valid. The committee are very grateful to Stefan Kottwitz for providing a new home for the FAQ.

At the same time as moving the current information, the committee are aware of the need to ensure that the FAQ will continue to be maintained and improved. This is too large a job to expect any one person to do it alone, and so we are seeking a small group of volunteers to help with the work. We have placed the source of the FAQ on GitHub (`https://github.com/uktug/uk-tex-faq`), to allow collaborative updating. Anyone wishing to contribute can do so, either by requesting access to the master repository, by making pull requests (experienced GitHub users will understand what this means), or by sending contributions by email (`faq@tex.ac.uk`).

We also intend to work on the website itself, and are hoping to provide a 'responsive' search facility (search as you type) and other modern facilities in the coming months.

As UK-TUG Secretary/Webmaster, I will be coordinating these efforts on behalf not only of the UK-TUG committee but also the wider TEX community. Assistance with any aspect of running the FAQ will be gratefully received.

References

Fairbairns, Robin. "A compilation of Frequently Asked Questions with answers". `http://ctan.org/pkg/uk-tex-faq`, 2014.

◇ Joseph Wright
Secretary UK TEX Users' Group
`secretary (at) uk.tug.org`

Appendix

As an indication of the breadth and depth of the current FAQ, here are the top-level headings:

1. The background
2. Documentation and help
3. Bits and pieces of (LA)TEX
4. Acquiring the software
5. TEX systems
6. DVI drivers and previewers
7. Support packages for TEX
8. Literate programming
9. Format conversions
10. Installing (LA)TEX files
11. Fonts
12. Hypertext and PDF
13. Graphics
14. Bibliographies and citations
15. Adjusting the typesetting
16. How do I do ...
17. Symbols, etc.
18. Macro programming
19. Things are going wrong ...
20. Why does it *do* that?
21. The joy of TEX errors
22. Current TEX-related projects
23. You're still stuck?

Single- and multi-letter identifiers in Unicode mathematics

Will Robertson

Abstract

This paper argues that separate spacing and therefore separate font invocations is needed for single- and multi-letter identifiers in mathematical contexts. This is not explicitly provided for by Unicode mathematics nor by the OpenType mathematics fonts that currently exist. The unicode-math package has been extended to accommodate these needs, which provides better compatibility with legacy LaTeX and amsmath documents.

1 Introduction

Unicode mathematics has been developing over the last fifteen years, spearheaded by Barbara Beeton [1, 2, 3], and is now approaching mainstream use. Ulrik Vieth has described OpenType mathematics and its relationship to TeX [6, 7]. Although work began on Unicode mathematics in XeTeX and LuaTeX several years prior [4], the unicode-math package was first formally released in 2010 [5].

Although much care was taken to incorporate as many mathematical alphabetic symbols into Unicode as possible, within the limitations imposed by the Unicode Technical Committee at the time, there is not a direct correspondence between the de facto mathematics alphabets defined in LaTeX and those used in Unicode. The original version of the unicode-math package did not take these differences into account, which has led to certain problems as Unicode-aware TeX engines have become more popular. This article will discuss the limitations of the original unicode-math package and the new interfaces set up to allow a smoother transition from legacy LaTeX documents to Unicode.

2 Brief overview

The unicode-math package builds on LaTeX's math font selection system and implements the entire symbol repertoire of Unicode mathematics (many thousands of glyphs) for direct use within LaTeX. As unicode-math requires the XeTeX or LuaTeX engine, glyphs can be inserted as Literal Unicode characters if desired. Control sequences are also defined in order to be able to access each symbol by name, based on the work of Barbara Beeton. Mathematical alphabet commands are also defined to emulate LaTeX's and amsmath's traditional `\mathbf`, `\mathfrak`, etc., commands. The difference with such alphabets in Unicode mathematics is that all glyphs come from within a single font, whereas in 8-bit TeX systems accessing these glyphs involved *switching* fonts of different styles.

$$\mathbf{G} + \boldsymbol{\Gamma} = G + \Gamma$$

```
\setmathfont{xits-math.otf}%
  [math-style=TeX]
```

```
$\symbf{G}+\symbf{\Gamma} = G+\Gamma$
```

Example 1: An example of 'TeX'-style mathematics.

$$\boldsymbol{G} + \boldsymbol{\Gamma} = G + \Gamma$$

```
\setmathfont{xits-math.otf}%
  [math-style=ISO]
```

```
$\symbf{G}+\symbf{\Gamma} = G+\Gamma$
```

Example 2: An example of 'ISO'-style mathematics.

A Unicode mathematics font is loaded with the command such as:

```
\setmathfont{xits-math.otf}
```

Optional arguments to the package or to the font loading command can be used to change the style of the mathematics; for example, publishers differ on whether Greek letters should be upright or italic by default, or whether bold Latin letters should be upright or italic. These cases are compared in Examples 1 and 2. These examples also show the use of the `\symbf` command to turn a single-letter alphabetic symbol 'bold' according to the prevailing style. This is a new command that replaces the use of `\mathbf` in previous versions of the package, for reasons that we will now discuss.

3 Single-letter vs. multi-letter symbols

Unicode mathematics defines a large number of mathematical alphabet symbol ranges, including alphabets of upright, italic, sans serif, fraktur, and script shapes in regular and bold weights, among others. Many of these support both Latin and Greek letters, plus a few assorted 'Greek-like' symbols such as `\nabla` (∇) and `\partial` (∂).

In legacy LaTeX and amsmath-based mathematics, the number of alphabets is more restricted, and in many cases commands such as `\mathbf` are used for both single-letter symbols and multi-letter identifies. In contrast, the design of Unicode mathematics fonts to date has including kerning around the alphabetic symbols suitable for single letter symbols only, and these symbols are often not appropriate to use in multi-letter contexts.

Examples are given following of mathematics in which single-letter and multi-letter identifiers are used in the different mathematical contexts for a range of alphabet styles.

3.1 Upright regular text

Upright regular-weight roman text is occasionally used in applied mathematics to represent constants such as e, the exponential:

$$\mathrm{e}^{-st} = \mathrm{e}^{-i\omega t}\,.$$

In some cases upright roman is used for the complex number ($\mathrm{i} = \sqrt{-1}$). Since these symbols may be used in contexts in which they appear adjacent to a separate symbol they should have additional spacing to separate them, such as in the (trivial) case of 'ee = e^2'. As another common example, the derivative operator $\frac{d}{dt}$ is often written $\frac{\mathrm{d}}{\mathrm{d}t}$.

In fluid dynamics in particular, dimensionless values such as the Reynolds number Re are often typeset using upright symbols:

$$\mathrm{Re} = \rho \mathbf{v} l/\mu\,.$$

Other dimensionless values include, for example, the Froude number (Fr) and the Strouhal number (St). Since these identifiers are essentially abbreviations of words and names, they should be kerned as text to bind the letters together visually.

3.2 Italic text

The aforementioned dimensionless numbers are also frequently typeset as italic text, instead of upright (roman) text:

$$Re = \rho \mathbf{v} l/\mu\,.$$

In such cases it is plain that there needs to be a clear distinction between adjacent symbols and the dimensionless number. For 'R' and 'e' there is little difference, but take the Froude number:

$$Fr \neq Fr$$

This is typeset with '`\mathit{Fr} \neq Fr`'. The extra kerning on the right hand side distinguishes the symbols as being separate.

Other examples from the literature include italic being used for operator-like notation, such as in these arbitrary cases taken from the literature:

$$\overline{Sep}_{\mathrm{I}}(\Sigma_1^{[C]})\,, \qquad \frac{\partial |det\, g|}{\partial g_{\mu\nu}}\,.$$

3.3 Bold upright text

Bold upright alphabetic symbols are commonly used in physics and engineering to denote vectors and matrices:

$$\mathbf{x} = [x,\, \dot{x},\, \theta,\, \dot{\theta}]^{\mathrm{T}}\,.$$

In mathematics, it is also common for bold upright roman to be used for both single *and* multi-letter abbreviations such as

$$DG = \hom(G, \mathbf{R}/\mathbf{Z}) \qquad \mathbf{Grp} \to \mathbf{Ab} \to \mathbf{Grp}$$

In the first case, the symbols should be kerned as separate glyphs, whereas in the second the letters should be spaced as in text.

Further examples could be given for sans serif regular weight and bold, for example. In contrast, alphabet ranges such as fraktur ($\mathfrak{ABC}$) and script ($\mathcal{ABC}$) are generally used for denoting single symbols only — although it is possible to find examples of notation such as

$$\mathit{Hom}(\mathscr{H}, \mathscr{E})\,.$$

The term 'Hom' is in a calligraphic style, whereas its arguments are shown in a 'curly' script style; Unicode mathematics, for now, only supports the latter, but unicode-math can be configured to support both if the font (such as STIX) contains glyphs for both.

The various examples listed above using single- and multi-letter identifiers cannot be represented using a single Unicode mathematics font, since each alphabetic range has a fixed spacing; in general, to date all Unicode mathematics fonts space the alphabetic glyphs as individual letters. Furthermore, when used in 'word-like' contexts, it is important to recognise that multilingual varieties of strings such as 'sin', 'cos', and 'tan' are possible, and that the fonts used to typeset such identifiers should support the entirety of Unicode, not just the restricted set of alphabetic symbols defined as 'mathematical'.

4 Additions to unicode-math

These factors lead to a definite tension between the commands defined originally in unicode-math, which mapped directly to Unicode mathematical alphabetic symbols only, and the commands that users expected from legacy LaTeX documents.

Due to the clear requirements of supporting single- and multi-letter identifiers, the approach now taken in unicode-math is to define a new set of commands, `\symbf`, `\symup`, `\symsf`, and so on, which switch to the Unicode mathematics *symbols* that can be used for single-letter identifiers. For multi-letter operators, LaTeX's traditional `\mathit`, `\mathbf`, etc., commands are retained as *font switches*, and do not perform any 'remapping' on their inputs. Fraktur and calligraphic alphabets, for example, are defined using `\symfrak` and `\symcal`, but familiar LaTeX commands `\mathfrak` and `\mathcal` are provided as synonyms for backwards compatibility.

Unless explicit fonts for `\mathbf` and so on are selected (see next), they are automatically selected

$\mathup{ηµ}(\omega) = y/\rho$

```
\setmainfont{Iwona-Regular.otf}
\setmathfont{texgyrepagella-math.otf}%
  [Scale=0.85]
```

```
$ \mathup{ηµ}(ω) = y/ρ $
```

Example 3: Greek text and symbols in mathematics.

$\rho_p = \text{Ш}_p * \rho$

```
\setmathfont{texgyrepagella-math.otf}%
  [Scale=0.85]
\setmathfontface\mathcyr{Charter Roman}
```

```
$ ρ_p = \mathcyr{Ш}_p \ast ρ $
```

Example 4: Cyrillic Sha being used in mathematics.

from the default text fonts. This allows, for example, the case in Greek schoolbooks in which the sine rule might be written as shown in Example 3. By the way, `\mathup` is defined in unicode-math as a synonym for `\mathrm` for exactly such situations; we're not always typesetting 'RoMan' text any more.

Restoring the idea of the '`\mathXYZ`' commands being 'text fonts in mathematics mode' also provided the opportunity to restore the functionality of LaTeX's `\DeclareMathAlphabet`, and to define a new unicode-math interface to it. It is now possible to write, say,

```
\setmathfontface\mathittt
  {texgyrecursor-italic.otf}
```

to define the math font command `\mathittt{...}`, which selects (in this case) an italic typewriter font. This command can also be used to select fonts for the built-in commands `\mathbf`, `\mathsf`, etc.

The `\setmathfontface` command also provides the possibility of selecting particular fonts for typesetting characters in other scripts. For example, the Cyrillic glyph 'Sha' is often used to typeset the Dirac Comb Function, and this can now be supported easily as shown in Example 4.

5 Compatibility

After upgrading to version 0.8, many users will wish to adapt their old documents written in unicode-math to the new syntax from `\mathbf` to `\symbf` and similar. Package options `mathbf=sym` will rename `\mathbf` to have behaviour as in previous version, with similar options for `mathsf` and so on. The legacy commands for switching to a text font in math mode are renamed to `\mathtextbf`, `\mathtextsf`, etc.

A common suggestion is to implement a 'smart' version of `\mathbf` that analyses the number of characters in its arguments. Writing `\mathbf{abc}` could switch to `\mathtextbf`, and `\mathbf{A}` to `\symbf`. This interface would be convenient for Latin-based examples, but further consideration is required before deciding it's actually a good idea.

In such a 'smart' version of the command, a case such as `\mathbf{\alpha\alpha}` would result in mathematics symbols inside a command that is intended for typesetting text, which would not produce expected output. (Exactly as in the case now in regular LaTeX when the unsuspecting user writes `\mathbf{\alpha}`.) In the 'smart' approach, having `\mathbf{\alpha}` work 'as expected' but then having `\mathbf{\alpha\alpha}` 'fail' does not seem like a sensible approach.

6 Conclusion

The unicode-math package now contains a full suite of commands to select alphabet styles suitable for both single- and multi-letter identifiers. For ideal typesetting purposes, developers of OpenType mathematics fonts should provide font features that allow selection of kerning suitable for both mathematics (single-letter) and 'text' (multi-letter) purposes.

References

[1] Barbara Beeton. Unicode and math, a combination whose time has come — Finally! *TUGboat*, 21(3):176–185, September 2000. http://tug.org/TUGboat/tb21-3/tb68beet.pdf.

[2] Barbara Beeton. The STIX project — from Unicode to fonts. *TUGboat*, 28(3):299–304, 2007. http://tug.org/TUGboat/tb28-3/tb90beet.pdf.

[3] Barbara Beeton, Asmus Freytag, and Murray Sargent III. Unicode support for mathematics. Technical report, Unicode, Inc., 2015. http://www.unicode.org/reports/tr25.

[4] Jonathan Kew. XeTeX live. *TUGboat*, 29(1):146–150, 2007. http://tug.org/TUGboat/tb29-1/tb91kew.pdf.

[5] Will Robertson. Unicode mathematics in LaTeX: Advantages and challenges. *TUGboat*, 31(2):211–220, 2010. http://tug.org/TUGboat/tb31-2/tb98robertson.pdf.

[6] Ulrik Vieth. Do we need a 'Cork' math font encoding? *TUGboat*, 29(3):426–434, 2008. http://tug.org/TUGboat/tb29-3/tb93vieth.pdf.

[7] Ulrik Vieth. OpenType math illuminated. *TUGboat*, 30(1):22–31, 2009. http://tug.org/TUGboat/tb30-1/tb94vieth.pdf.

◇ Will Robertson
The University of Adelaide, SA, Australia
will dot robertson (at) latex-project dot org

Trilingual templates for an educational institute in Bashkortostan, Russia

Boris Veytsman and Leyla Akhmadeeva

Abstract

Creation of document styles for an organization that uses a non-Western script is always a challenge: the organization needs to support both Western and non-Western elements in its documents. The new Institute of Continuous Professional Medical Development in Ufa poses a special challenge, because we want its templates to be trilingual, with English, Russian and Bashkir elements. The Bashkir language uses a Cyrillic script, which is close to but different from Russian Cyrillic.

1 Introduction

A medical professional must constantly update her knowledge and skills, keeping current with the recent advances in the field. In Russia, as in most other countries, a continuing education is one of the preconditions for medical license renewal.

The required education may be provided not only by state-run medical universities, but also by private institutes. The latter are rather new in the Russian medical system.

The Institute of Continuous Professional Medical Development in Ufa is a private educational enterprise created in February 2015 by the Bashkortostan National Health Chamber [1]. We were asked to create document and presentation templates for it.

This is an interesting challenge because Bashkortostan has two official languages: Russian and Bashkir. They both use a Cyrillic script, but Bashkir has nine letters absent in Russian. For international letters we want to add an English header. Thus we needed to combine different scripts and blend them into common documents.

2 An aside: Symbolism and logo

We asked a prominent Ufa artist and specialist in heraldry, Airat Usmanov, to compose a logo for the institute, asking that the logo reflect the symbolism of the Bashkortostan republic as well as Russia and Bashkortostan National Health Chamber.

The symbols of the Bashkortostan Republic are shown in Figure 1. They use white, blue, green and gold colors. The Russian flag is red, blue and white. The artist skillfully combined these motifs in the logo (Figure 2).

3 Documents and letters

A Russian national standard [2] establishes rather strict rules about the elements of an official document and their positioning. Of particular interest are rules for multilingual documents. They require the elements of the headers in two different languages to be on the same level. Presumably this implies that the fonts for the headers should be compatible and have the same sizes.

We chose the ParaType fonts [3] as having a full set of Latin and Cyrillic letters and being very legible. The fact that we had experience working with ParaType [4–6] also influenced our decision.

The official standard [2] describes documents with the headers in two languages, but not three. Therefore we decided to make two options: Russian-Bashkir headers for domestic documents, and Russian-English for international ones. The ParaType fonts provided an excellent blending of the headers, as seen in Figures 3 and 4.

One problem with Bashkir language support in TEX is the lack of corresponding hyphenation patterns. Therefore our package provides only a limited support for the Bashkir language. Fortunately, document headers are not hyphenated.

Since we wanted the package to work for any input encoding, we used a well-known trick of letting Cyrillic letters be macros, so the headers were written in the following rather cumbersome way:

```
<<\CYRSHHA\CYRA\CYRU\CYRL\CYRERY\CYRKBEAK{}
\CYRSHHA\CYRA\CYRKBEAK\CYRL\CYRA\CYRU{}
\CYROTLD\CYRL\CYRK\CYRSCHWA\CYRSHHA\CYRE
\CYRN\CYRD\CYRSCHWA\CYRG\CYRE{}\\
\CYROTLD\CYRZDSC\CYRL\CYROTLD\CYRK\CYRSHHA
\CYROTLD\CYRZDSC{}
\CYRSHHA\CYROTLD\CYRN\CYRSCHWA\CYRR\CYRI{}
\CYRB\CYRE\CYRL\CYRE\CYRM{} \CYRB\CYRI
\CYRR\CYRE\CYRY{} \CYRI\CYRN\CYRS\CYRT\CYRI
\CYRT\CYRU\CYRT\CYRERY>>
```

4 Beamer theme

We based the presentation template on the Beamer sidebar theme [7]. We used the colors of the Bashkir flag and the logo. To keep the integrity of the Institute typographic image, we used the sans serif version of the ParaType fonts. The examples are shown in Figure 5.

5 Conclusions

We found that a formal document provides interesting challenges to a style designer. Multilingual typesetting makes the work even more challenging. Fortunately, TEX is a good tool to solve these problems.

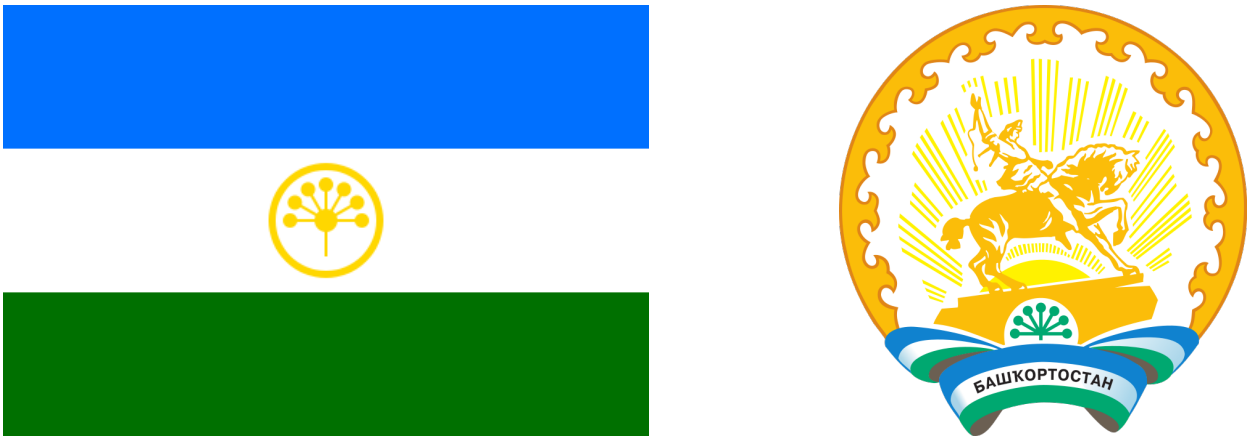


Figure 1: Bashkortostan flag and coat of arms

Figure 2: Institute logo by Airat Usmanov

References

[1] Institute of continuing professional education for healthcare practitioners website. `http://nmp-rb.ru/http-/nmp-rb.ru/http-/nmp-rb.ru/institut`, 2015.

[2] Gosstandard, Russia. *GOST R 6-30-2003. Unified systems of documentation. Unified system of managerial documentation. Requirements for presentation of documents*, 2003.

[3] Pavel Farář. *Support package for free fonts by ParaType*, February 2014. `http://ctan.org/pkg/paratype`.

[4] Leyla Akhmadeeva, Ilnar Tukhvatullin, and Boris Veytsman. Do serifs help in comprehension of printed text? An experiment with Cyrillic readers. *Vision Research*, 65:21–24, 2012.

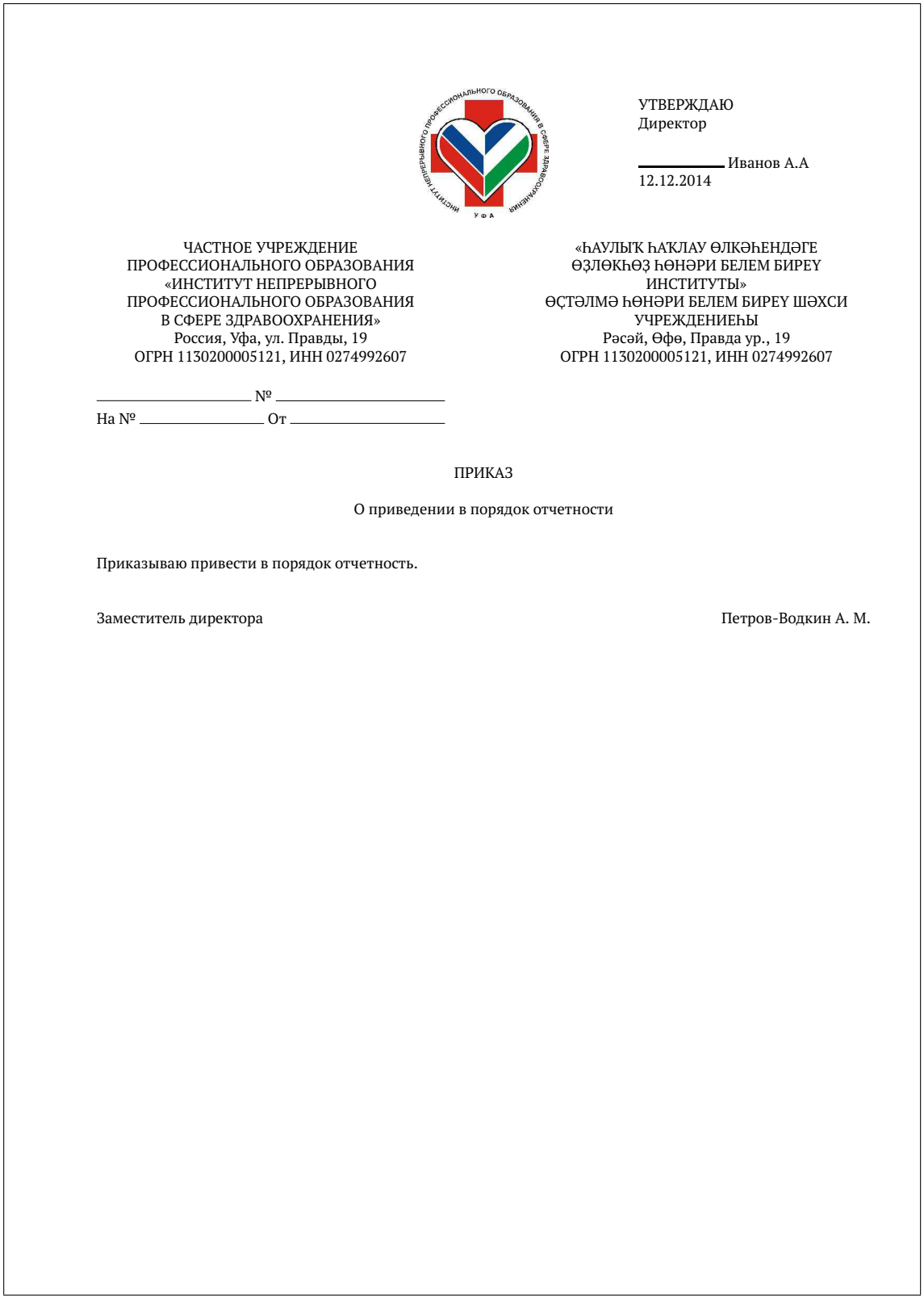
[5] Boris Veytsman and Leyla Akhmadeeva. Towards evidence-based typography: First results. *TUGboat*, 33(2):156–157, 2012. `http://tug.org//TUGboat/tb33-2/tb104veytsman-typo.pdf`.

[6] Leyla Akhmadeeva and Boris Veytsman. Typography and readability: An experiment with post-stroke patients. *TUGboat*, 35(2):195–197, 2014. `http://tug.org/TUGboat/tb35-2/tb110akhmadeeva.pdf`.

[7] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*, March 2015. `http://ctan.org/pkg/beamer`.

◇ Boris Veytsman
Systems Biology School and Computational Materials Science Center
MS 6A2
George Mason University
Fairfax, VA 22030 USA
`borisv (at) lk.net`
`http://borisv.lk.net`

◇ Leyla Akhmadeeva
Bashkir State Medical University, 3 Lenina Str., Ufa, 450000, Russia
`la (at) ufaneuro (dot) org`
`http://www.ufaneuro.org`



УТВЕРЖДАЮ
Директор

_____ Иванов А.А
12.12.2014

ЧАСТНОЕ УЧРЕЖДЕНИЕ
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ИНСТИТУТ НЕПРЕРЫВНОГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
В СФЕРЕ ЗДРАВООХРАНЕНИЯ»
Россия, Уфа, ул. Правды, 19
ОГРН 1130200005121, ИНН 0274992607

«ҺАУЛЫҠ ҺАҠЛАУ ӨЛКӘҺЕНДӘГЕ
ӨҘЛӨКҺӨҘ ҺӨНӘРИ БЕЛЕМ БИРЕҮ
ИНСТИТУТЫ»
ӨҪТӘЛМӘ ҺӨНӘРИ БЕЛЕМ БИРЕҮ ШӘХСИ
УЧРЕЖДЕНИЕҺЫ
Рәсәй, Өфө, Правда ур., 19
ОГРН 1130200005121, ИНН 0274992607

_____ № _____
На № _____ От _____

ПРИКАЗ

О приведении в порядок отчетности

Приказываю привести в порядок отчетность.

Заместитель директора Петров-Водкин А. М.

Figure 3: A document with Russian and Bashkir headers

ЧАСТНОЕ УЧРЕЖДЕНИЕ
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ИНСТИТУТ НЕПРЕРЫВНОГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
В СФЕРЕ ЗДРАВООХРАНЕНИЯ»
Россия, Уфа, ул. Правды, 19
ОГРН 1130200005121, ИНН 0274992607

PRIVATE INSTITUTE
OF PROFESSIONAL EDUCATION
“INSTITUTE OF CONTINUING
PROFESSIONAL EDUCATION
FOR HEALTHCARE PRACTITIONERS”
Russia, Ufa, Pravda St., 19
OGRN 1130200005121, INN 0274992607

_____ № _____
На № _____ От _____

TEX Users Group
PO Box 2311
Portland, OR 97208-2311
USA

Dear Friends,

Many thanks for your help with the travel to Darmstadt. It was a great conference!

Yours,

Leyla Akhmadeeva

Figure 4: A letter with Russian and English headers

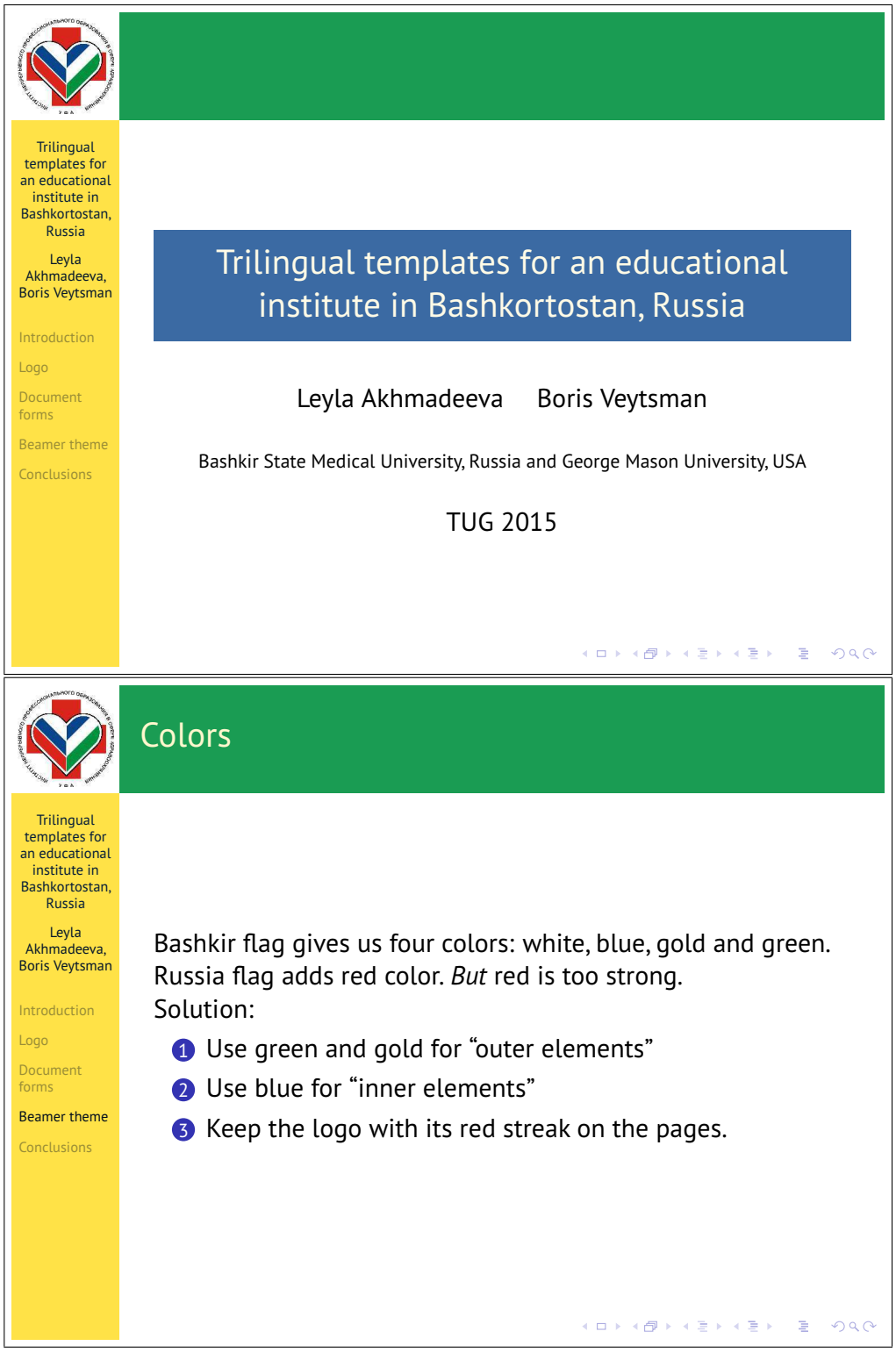


Figure 5: Beamer theme examples

Joseph's Adventures in Unicodeland

Joseph Wright

1 Unicodeland

The rich variety of human language has over time led us to a plethora of ways of writing down our communications. Historically, computer systems handled this poorly. The 26 letters of the English alphabet have defined the landscape of computerized text storage, first through the ASCII standard and later as the basis of many 8-bit systems. Over roughly the past quarter-century, this situation has begun to change with the development of the Unicode standard (The Unicode Consortium, 2015). Unicode takes us beyond the limitations of the 7- or 8-bit world into a much richer environment. In this 'Unicodeland' every character has its proper place, and every character can and should be correctly handled by compliant software.

The Unicode Consortium have defined not only a rich (and expanding) set of characters (or more accurately *code points*) to handle this variety of data, but have also explored and defined how these should be manipulated under a range of transformations. As befits a standards organisation, the Unicode Consortium have not tied this information to any particular implementation. Rather, they provide a set of machine-readable files and documentation guidelines to allow compliant implementations to be built for the range of tasks we use computers for.

In the TEX world, we have today two Unicode-capable engines in general use: XETEX and LuaTEX. Following a careful TEX tradition, these engines do not hard-code Unicode data or behaviours into the engines. Rather, they allow us to handle Unicode input and to control behaviours by reading in the appropriate data.

Here, I will look at two areas where we need to get Unicode data into the engine: setting up TEX's codes correctly for the Unicode range, and implementing case-changing. Whilst the focus here is on how we are tackling these problems for LATEX, the ideas should apply to all TEX users.

2 Setting up characters

As XETEX and LuaTEX can accept input across the full Unicode range they need to know how to treat a much greater number of characters than classical TEX engines. For example, with an 8-bit engine we usually restrict 'letters' for creating control word names to `A`–`Za`–`z`. With a Unicode engine that's not reasonable: anything that is a letter according to the agreed standard can and should be set to category code 11. However, what we don't want to do is code all of that in by hand. Luckily, all of the core Unicode data files are provided in plain text format (and indeed are written in ASCII) and are machine-readable. 'All' we have to do is parse the appropriate files as part of the TEX run.

The details of course are a little more complex. It turns out that we want to set up several things

- `\catcode`
- `\lccode`
- `\uccode`
- `\Umathcode`
- `\XeTexintercharclass` (XETEX only)

for all appropriate characters. To do that, we need to first work out which Unicode data files have the relevant information in them and then to parse it into a usable form.

As the Unicode Consortium deal with data for many purposes, it is not surprising that things like TEX's `\catcode` concept don't feature directly in the data files. Instead, we need to make some systematic decisions about relating Unicode properties to TEX. Most of this work was done by Jonathan Kew when he first released XETEX; at that time, he created a Perl script (`unicode-char-prep.pl`) to do the work (Kew, 2015). The LATEX team has now created a very similar script, using pdfTEX rather than Perl for the parsing but retaining most of the logic.

Much of the conversion is relatively obvious. Thus for example characters described by the Unicode Consortium as falling into one of the letter types are mapped to `\catcode` 11. However, there are other characters that need to be `\catcode` 11: combining marks and East Asian ideographs. Picking these up requires a bit of thought: the details of parsing the source files are more technical than conceptual. Reading all of the data with pdfTEX takes a few seconds, but luckily this only has to happen on the machine of one of the members of the team. For users, the processed file can be read *very* quickly: indeed, as part of format-building, it's negligible.

What the team has added in this area is setting up a single file to be read by both XETEX and LuaTEX, with the necessary conditionals inside the file. That means that the common outcomes are the same in all cases, with just the *additive* part for XETEX covering `\XeTexintercharclass`. Having the file provided by the team also means that it will be updated as part of wider kernel changes, which should provide some regularity as to when this takes place.

3 Case changing

3.1 The background

TEX provides us with two primitives for case changing, `\lowercase` and `\uppercase`. The logic behind these is simple: they convert single characters from one case to another based on the idea of one-to-one relationships. That's fine when we have a simple situation with two cases, one language and everything mapping neatly. This is, of course, TEX's background: for English, `\lowercase` and `\uppercase` are entirely reasonable.

Life gets more complicated once we introduce more variation. First, even apparently simple one-to-one relationships can be language-dependent. Perhaps the most obvious example is Turkish, where the upper case equivalent of `i` is `İ`, not `I`. Second, there's no context-dependence available: mappings are not always the same for the same characters. The Greek 'final sigma' rule is perhaps the best known of these situations: the correct lower casing of `ΌΔΥΣΣΕΎΣ` for example is `όδυσσεύς`, using the two different lower case sigma characters in Greek. There is then the question of the one-to-one mappings themselves: `fußball` in upper case is `FUSSBALL` with an extra character. It's clear from these issues (and other subtleties) that a more nuanced approach is needed.

In practice, making everything work with Unicode input requires a Unicode engine, so the ideas here work fully only with XeTEX and LuaTEX. With pdfTEX, the best fall-back is to cover just the ASCII range. Unlike the first part of this article, here we are also discussing code for LATEX3, thus at the expl3 programming level (The LATEX3 Project, 2015). The commands therefore have 'real' names that might seem unusual: to avoid obscuring the ideas, I'll give them 'design' (CamelCase) names in the examples here.

3.2 The approach

The first thing to recognise is that when we want to talk about case changing, we are talking about *text*. There may be some embedded formatting to skip (more on that in a bit), but we can work on the basis that we are case-changing category code 11 and 12 tokens. Of course the TEX primitives have important uses in generating 'funny' tokens (as they change character codes but not category codes): that's got essentially nothing to do with the case of characters at all!

With a bit of effort it's possible to set up an expandable loop over a list of tokens that preserves all of the spaces and brace groups. Using that approach, we can pull out tokens one by one for conversion. Spaces are passed straight through, while we need to use a recursive approach with groups. So this leaves the question of dealing with 'normal' tokens.

Converting each token is done by using a look-up table made up of 100 control sequences, each covering part of the full Unicode range. This approach offers a balance between efficiency and performance. Using a table of this form, we are not limited to one-to-one look-ups: one-to-many is also available. This core idea covers a large part of the situations we need, but to get the context dependence needs some specialised code. In the current approach, that is done using look-ahead routines dedicated to each special situation.

Covering language-dependent mappings needs a version of the code that tracks the currently-active language. The number of special cases we find for this is pretty small, so each one can then be handled using some custom code.

3.3 Features

The key features of the case changer are those related to the Unicode standards: the one-to-one mappings and more complex relationships follow those given by the consortium. The basics are built-in: detection of context and providing a way to indicate the language of the text as an additional argument to case changing. Thus

```
\edef\test{%
  \ExplLowerCase{RAGIP HULÛSİ}%
}
\show\test
```

yields

```
> \test=macro:
->ragip hulûsi̇.
```

whilst

```
\edef\test{%
  \ExplLowerCase[tr]{RAGIP HULÛSİ}%
}
\show\test
```

yields

```
> \test=macro:
->ragıp hulûsi.
```

We can see that in the second case we get not only full mapping of the Unicode characters but also the difference in treatment of the dotted and dotless I.

From a programmer's point of view, it's convenient to be able to do case changing expandably. As we can see in the examples above, that's exactly what the code offers: the case changer can be used inside an `\edef`. That means we can easily store

the 'real' case changed text without having to jump through any TEX primitive hoops.

At the user level, some things should be skipped by case changing: math mode material and explicitly-marked input. Following the pattern set up by the textcase package (Carlisle, 2004), these are handled by detecting $ (etc.) and using a dedicated 'opt-out' command, respectively. Thus we obtain

```
\edef\test{%
  \ExplUpperCase
    {Some maths $y = mx + c$}%
}
\show\text
...
> \test=macro:
->SOME MATHS $y = mx + c$.
```

and

```
\edef\test{%
  \ExplUpperCase
    \NoChangeCase{FeFe}-hydrogenase}%
}
\show\text
...
> \test=macro:
->\NoChangeCase {FeFe}-HYDROGENASE.
```

One of the subtle features of Unicode case changing is what they call titlecasing. This is the process whereby the first character of a piece of text is made upper case, with the rest being lower case. The subtle part is that a few characters need special handling if they are first: these tend to be situations where the single glyph looks a bit like two letters. We've called this "mixed case": titlecasing in English at least implies some form of word-level processing.

```
\edef\test{%
  \ExplMixedCase{ǆ}%
}
\show\text
...
> \test=macro:
->ǅ.
```

Perhaps the best example of this behaviour is with the combination IJ in Dutch.

```
\edef\test{%
  \ExplMixedCase{ijsselmeer}%
}
\show\text
...
> \test=macro:
->Ijsselmeer.
\edef\test{%
  \ExplMixedCase[nl]{ijsselmeer}%
}
\show\text
...
> \test=macro:
->IJsselmeer.
```

The final area to consider is *case folding*. This looks very much like lower casing but it's not meant for text: it's a process for programmers. Case folding is a strictly one to one mapping with no context dependence. We've provided this using a simplified approach (no special tests), and based on yet another Unicode data file.

4 Conclusions

Using Unicode data in TEX needs a bit of thought to match up the ideas of the two systems. However, we can do that and benefit from the tremendous amount of work done by the Unicode Consortium. In return, we enable users to get predictable outcomes from their code and to match up with the handling of other computational systems. This will only become more important in the future.

Thanks to Jonathan Kew for creating the Perl script used as a basis for our Unicode data parser. Thanks also to Bruno Le Floch who developed the looping approach used for case changing and the method for compacting the data into an efficient format.

References

Carlisle, David. "The textcase package". Available on CTAN: macros/latex/contrib/textcase, 2004.

Kew, Jonathan. "XeTeX". http://xetex.sourceforge.net/, 2015.

The LaTeX3 Project. "The expl3 package". Available on CTAN: macros/latex/contrib/l3kernel, 2015.

The Unicode Consortium. "The Unicode Standard". http://www.unicode.org/versions/latest/, 2015.

◇ Joseph Wright
Morning Star
2, Dowthorpe End
Earls Barton
Northampton NN6 0NH
United Kingdom
joseph.wright (at)
morningstar2.co.uk

Through the \parshape, and what Joseph found there

Joseph Wright

1 Paragraph shape

The shape of the paragraph defines how text looks on the page. To talk about paragraph shape, we first need to think about how it relates both to the text itself and to the outer 'container' in which we are placing the paragraph (see Figure 1). Text in paragraphs is placed in vertical containers, 'galleys', which are themselves then placed on the page. The galley edges (thick black lines in the figure) may be separated by a margin from the edges of the paragraph (the light grey box in the figure). The paragraph shape itself may be a simple rectangle, as illustrated, or may be a more complex shape, as we will see below. Within that shape, the text itself is placed on a line-by-line basis. Not all of those lines of text will necessarily use the full width of the paragraph shape: the last line is often ended short of the margin, while in many styles the first line of a paragraph has a marker indent.

TEX provides us with a variety of primitives which are in some way linked to the shape of a paragraph. *TEX by Topic* (Eijkhout, 1992) lists seven primitives in the 'Paragraph Shape' chapter:

- \parindent
- \hsize
- \leftskip
- \rightskip
- \hangindent
- \hangafter
- \parshape

The \parindent primitive can be thought of as controlling appearance *within* the paragraph shape, whilst \hsize controls what the shape has to fit within. Both \leftskip and \rightskip are linked more to justification than to paragraph shape. This leaves \hangindent, \hangafter and \parshape to set up the shape of the paragraph itself. And the two \hang... primitives are in fact special cases of \parshape, so if we want to think about paragraph shape in TEX primitive terms we must focus on \parshape.

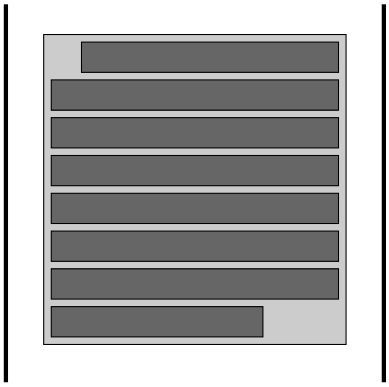


Figure 1: Paragraph shape

2 Views of \parshape

Matching up the TEX view of paragraph shape with the ways we can think about design requires us to think carefully about manipulating \parshape. Different design elements interact; thus, creating complex layouts by hand is time-consuming. Rather than do this, an alternative approach is to conceptualise these different design aspects and to provide interfaces (and data structures) for each of them. We can then construct the necessary \parshape programmatically, freeing us to describe design in a more natural way (at the cost of the effort in creating the underlying logic).

As part of the experimental LATEX3 galley module (The LATEX3 Project, 2015), the team has been exploring how we can achieve this separation. Some of the concepts are easy to implement, whilst others are more challenging, particularly given the nature of the underlying TEX model. Here, I will survey the design concepts we have identified and how we have currently tackled each one. The focus is very much on ideas rather than code: for the latter, readers are encouraged to consult l3galley.pdf and xgalley.pdf, both available on CTAN.

2.1 Margins

The paragraph shape will have left and right margins separating it from the galley edges. (These margins may of course be of zero length.) This is by far the most straightforward part of the design description of a paragraph. We can add margins to a paragraph either by specifying the absolute distance from the edge of the galley, or by describing a margin relative to any existing margin. An interface for both of these requires only three data items

- The left margin
- The right margin
- Whether to apply these on a relative or absolute basis

A suitable \parshape can then be created to implement these requirements: existing margins can be tracked separately or can be recovered from the existing \parshape using the ε-TEX \parshapeindent and \parshapelength primitives.

2.2 Shapes

Within the margins, the next design concept we can identify is the most obvious one of all: an actual shape applying to the paragraph. Such shapes are commonly seen in lists, which may use either a first-indented or first-hanging design. To allow maximum flexibility in this area it is useful to minimise the number of fixed decisions, which leads to an interface which requires

- A number of 'normal' (unmodified) lines
- A list of indents from the left margin
- A list of indents from the right margin
- A flag to indicate if the normal margins should resume after the last modified line

2.3 Cutouts

In contrast to margins and fixed paragraph shapes, which typically apply to a particular part of a document (for example, all quotations, all headers, and so on), the third view of paragraph shape is used on a one-off basis. A 'cutout' is a section of the paragraph which is (normally) indented to allow space for the insertion of an independent element: almost always a figure, with the text wrapping around it.

The current interface for this element is based on

- The side of the paragraph to cut
- The number of normal lines to leave
- A list of indents to apply to altered lines

In contrast to shaping a paragraph, there is no question here that the normal line length will resume: a cutout applies to a strictly fixed number of lines.

2.4 Combinations

On their own, each of the three different design views for `\parshape` are clear. The challenge at a code level is allowing fluid combinations of one or more of them to occur without the user needing to know that they are implemented using a single primitive.

By separating out the design elements and more importantly by tracking them in appropriate data structures, this is achievable. Notably, whilst the expectation is that margins and shapes obey TEX groupings (in LATEX terms, they are tied to environments), cutout parts need to act globally within the galley they apply to.

3 Challenges

By far the most challenging concept in the design of paragraph shape is handling cutouts in a 'complete' sense. These constructs are unused in many document designs but where they are used, they throw up a wide range of issues.

The `\parshape` primitive is a rare example of a grid-like approach to typesetting in the TEX engine. The primitive works in terms of lines of text, an approach which only makes complete sense if the baseline-to-baseline distance is known. For simple designs this will be no issue, but once a rich mix of display-like elements (maths, headings, *etc.*) is included, creating cutout shapes which work reliably becomes much more challenging. At the same time, describing cutout elements is likely more naturally done using distances than numbers of lines: 'leave a space for a figure which is 5 cm high', for example. Accommodating these more complex design descriptions requires both more code and, more importantly, a more thorough focus on user expectations.

The link between `\parshape` and number of lines also shows in the fact that we can talk about unaltered lines at the start of a paragraph but not at the end. A short consideration of the implementation shows why this is: to deal with a paragraph 'bottom up' means breaking the lines to some fixed length, then finding if any of the lengths need altering, then re-breaking, *etc.* This contrasts with the 'top down' algorithm we have in TEX: for each line to break, we know the length allowed as part of the first pass.

Cutout parts are usually used in cases where they need to be applied as a single block: it is no good leaving part of the space for a wrapped figure at the bottom of one page and the remainder at the top of the next! Handling this requires some information from the page-breaking system, and a definition of a cutout which includes a floating element.

However, the biggest single challenge in using the new code is that `xgalley` requires full control of the `\par` primitive, and in manipulating `\parshape` we require that no other code modifies it. As such, whilst at present the `xgalley` code works well in controlled tests, it is likely to break badly with 'real life' documents. This is an area we are currently addressing.

References

Eijkhout, Victor. *TEX by Topic*. Addison-Wesley, Wokingham, United Kingdom, 1992. `http://www.eijkhout.net/texbytopic`.

The LATEX3 Project. "The `xgalley` package". Available on CTAN: `macros/latex/l3experimental/xgalley`, 2015.

◇ Joseph Wright
Morning Star
2, Dowthorpe End, Earls Barton
Northampton NN6 0NH UK
`joseph.wright (at) morningstar2.co.uk`

TEX and controlled access to information

Boris Veytsman

Abstract

While we in the TEX community strive to make information open, there are cases when controlling access to information is legitimate. We do not want to publish our passwords, medical histories or other sensitive details. Sometimes the information is not confidential, but different audiences can have different needs: consider students' vs. teachers' versions of a textbook.

There are two aspects of this problem. Output-level control means that we have a single source which can produce different output files depending on compilation options. Source-level control means having different versions of "sources" obtained from the same master file.

In this paper we discuss tools for both of these approaches and their implementation in a TEX system.

1 Introduction

One popular technology activist slogan is "Information wants to be free", attributed to Stewart Brand [1]. It seems almost sacrilegious to use free (as in speech) tools like TEX and friends to hide information. However, this slogan as stated should refer to scientific, technical or cultural information only: obviously there is plenty of private information that we do not want to be "free". For instance, GNU cryptography tools of high quality do exist.

There are several cases when information hiding may be desired. First, the information can be uninteresting. Imagine you create a formal statement of work for a software development project. This statement includes financial information (how much each part of the project costs), technical information (which libraries and languages will be used), project milestones, etc. One can imagine several audiences with different needs. Some (e.g., executives) would want to read the document in its entirety. The financial team may not want to be bothered by technical details, while the development team may be bored by the financial details. Therefore we need several overlapping versions of the output intended for the different audiences. We will call this *output level control*.

A related, but different problem arises when we have several authors, and want to hide parts of the document from some of its authors. This possibility may seem rather exotic, but consider a report having classified and unclassified information. Among its authors could be experts that lack the clearance to see some parts of the document. Public and secret parts could be interspersed: for example, we might have classified footnotes to the unclassified body text. We want the authors to be able to work on their parts without endangering the overall secrecy level. We will call this *input level control*.

In this paper we discuss both of these issues and our solutions.

2 Existing LATEX solutions for output level control

LATEX provides a number of options to control the output of the document. The most direct one is the `\include` command and the `\includeonly` mechanism. We can separate text into parts intended for different audiences, and for each audience use only the relevant parts.

However, each included part starts a new page in the document, which does not allow interspersed and overlapping parts intended for different audiences. Also, putting different parts in different files might be confusing, especially when there are more than two different audiences.

Some of these problems can be eliminated with the `\input` command, which does not start a new page. There is no `\inputonly` command in LATEX analogous to `\includeonly`, but it is easy to emulate it, for example,

```
\newif\iffinancial
...
\iffinancial\input{cost_table.tex}\fi
```

Still, the requirement to store content in different files is onerous.

The *comment* package [2] eliminates many of these problems. It allows the user to define comment-like environments and selectively output them with `\includecomment` and `\excludecomment` commands.

A similar approach is used by the *beamer* package [3], where one can include or exclude notes for presentations.[1]

When the package described in the next section was published on CTAN, Robin "ypid" Schneider informed me that another package with the same functionality, *tagging* by Brent Longborough, already exists [4]. I regret to say I simply missed this package in my search.

3 Output level control: a new package *multiaudience*

The new package *multiaudience* [5] tries to provide

[1] I am grateful to Joseph Wright and other participants of TUG'15 for this remark.

a clean interface for output level control. Its main concept is *audience*. A document can define several audiences using the `\SetNewAudience` macro, for example,

```
\SetNewAudience{admins}
\SetNewAudience{devs}
\SetNewAudience{execs}
```

This code defines three audiences, *admins*, *devs*, and *execs*.

The document can have one and only one *current audience*, which is stored in the `\CurrentAudience` macro. The author may define it using the command `\DefCurentAudience`

```
\DefCurrentAudience{admins}
```

or with just a TEX `\def`:

```
\def\CurrentAudience{admins}
```

The latter possibility allows the user to define the current audience outside the document (from the command line), for example

```
pdflatex -jobname devs \
 '\def\CurrentAudience{devs}\input{master.tex}'
```

This approach has the advantage of keeping the `tex` file (`master.tex` in this case) clean, and generating different versions on the fly.

The heart of the package is `\showto` macro. It has two arguments: the comma-separated list of audiences and the text to show to these audiences, for example,

```
\showto{admins}{This text
  is visible to admins only.}
```

The command can be nested:

```
\showto{admins, devs}{This text is visible
  to admins and devs. \showto{devs}{This
  text is visible to devs only.} This text
  is visible to admins and devs again.}
```

In the example above the text is visible to admins and devs with the exception of the italicized text, which is visible to devs only.

A variant of the `\showto` macro uses exclusion rather than inclusion logic: if the first argument starts with a minus sign, it defines the audiences which will *not* see the information:

```
\showto{-, execs}{This text is  visible to
  everybody but execs.}
```

The package provides the environment `shownto` with the similar syntax and semantics:

```
\begin{shownto}{admins,devs}
  This text is visible to admins and devs
  \begin{shownto}{-, admins}
    This text is visible to devs only
  \end{shownto}
  This text is visible to admins and
  devs again.
\end{shownto}
```

There are also special commands like `\Footnote`, making selectively visible footnotes:

```
We have a special footnote
command.\Footnote{admins}{This
footnote is for admins only.}
```

and section-like environments:

```
\begin{Subsection}{admins}[Short title]{Long
    title}
This subsection is visible only to admins
\end{Subsection}
```

Moreover, the user can define new commands and section-like environments with a simple interface:

```
\DefMultiaudienceCommand{\Footnote}%
   {\footnote}
\NewMultiaudienceSectionEnv{Subsection}%
   {\subsection}
```

The current implementation is very simple: basically we evaluate a TEX boolean `\if@MULTAU@shown` using the first argument of the `\showto` command, and typeset or not the second argument with the construction

```
\if@MULTAU@shown#2\fi
```

The actual implementation is slightly more involved since we need to check whether the first argument is "`-`", and accordingly to select inclusion or exclusion logic. See the source code [5] for the full details.

Petr Olšák wrote a plain TEX implementation even before my talk at TUG'15 was finished [6].

Of course this simplicity has its drawbacks: you cannot use verbatim construction in the second argument of `\showto` command. There are a number of workarounds here, see, e.g. [7].

4 Source level control

Source level control should solve two problems, one being easy, and another being slightly more difficult:

1. Extracting the material intended for different authors.
2. Accommodating the changes made by any author.

The second problem involves the following situation. Suppose a non-cleared author makes a change to the unclassified part of a document. We need to be able to accommodate her changes in the master document, which includes classified material.

Perl script *srcredact* [8] solves both these problems. Its interface is a simplified interface of the *docstrip* program [9]: we have special comment lines in

```
\documentclass{article}
\begin{document}
\title{A Letter to the Secretary
  of the Treasury}
\author{Mark Twain}
\date{Riverdale-on-the-Hudson, October 15, 1902}
\maketitle

%</ALL>
%<*uppercase|nobonds>
THE HON. THE SECRETARY OF THE TREASURY,
WASHINGTON, D.~C.:
%<*ALL>
%</uppercase|nobonds>
\textsc{the hon. the secretary of the treasury,
  washington, d.~c.:}
%<*ALL>

Sir,---Prices for the customary kinds of winter
fuel having reached an altitude which puts them
out of the reach of literary persons in
straitened  circumstances, I desire to place
with you the following order:

%</nobonds>
Forty-five tons best old dry government bonds,
suitable for furnace, gold 7 per cents.,
1864, preferred.
%<*ALL>

Twelve tons early greenbacks, range size,
suitable for cooking.

Eight barrels seasoned 25 and 50 cent postal
currency, vintage of 1866, eligible for
kindlings.

Please deliver with all convenient despatch
at my house in Riverdale at lowest rates for
spot  cash, and send bill to

Your obliged servant,

Mark Twain, Who will be very grateful,
and will vote right.
\end{document}
```

Figure 1: Example of a TEX file for *srcredact* tool

the TEX file (*guards*): either `%<*name1|name2|...>` or `%</name1|name2|...>`. The first one switches on the inclusion of text, while the second one switches it off. The special name `ALL` stands for all names. By default the text is considered to be enclosed in an `<*ALL>`/`</ALL>` pair.

An example input file is shown in Figure 1. It defines three versions of the same document, `default`, `uppercase` and `nobonds`. We can extract these three versions. If one version, say, `nobonds`, is changed, we can incorporate the changes into the main file and re-generate the three versions, as shown in Figure 2.

5 Conclusions

Information separation, hiding and control provide an interesting problem for document creation tools. TEX, being a programmable tool based on a text interface, is quite useful for solving it.

Acknowledgements

This work was partially supported by Neadwerx, Inc. and the US Consumer Financial Protection Bureau.

References

[1] R. Polk Wagner. Information wants to be free: Intellectual property and the mythologies of control. *Columbia Law Rev.*, 103:995–1034, 2003.

[2] Victor Eijkhout. *The comment package*, October 1999. `http://www.ctan.org/pkg/comment`.

[3] Till Tantau, Joseph Wright, and Vedran Miletić. *The beamer class*, March 2015. `http://www.ctan.org/pkg/beamer`.

[4] Brent Longborough. *tagging.sty. A package for document configuration*, August 2011. `http://www.ctan.org/pkg/tagging`.

[5] Boris Veytsman. *Generating multiple versions of a document for different audiences from the same source*, August 2015. `http://www.ctan.org/pkg/multiaudience`.

[6] Petr Olšák. Hidden text from unprivileged readers. `http://petr.olsak.net/opmac-tricks-e.html#showif`, July 2015.

[7] Timothy Van Zandt, Denis Girou, Sebastian Rahtz, and Herbert Voß. *The fancyvrb package. Fancy Verbatims in LaTeX*, May 2010. `http://www.ctan.org/pkg/fancyvrb`.

[8] Boris Veytsman. *A tool for redacting the sources*, August 2015. `http://www.ctan.org/pkg/srcredact`.

[9] Frank Mittelbach, Denys Duchier, Johannes Braams, Marcin Woliński, and Mark Wooding. *The DocStrip program*, November 2014. `http://www.ctan.org/pkg/docstrip`.

◇ Boris Veytsman
Systems Biology School and
Computational Materials
Science Center, MS 6A2
George Mason University
Fairfax, VA 22030 USA
`borisv (at) lk dot net`
`http://borisv.lk.net`

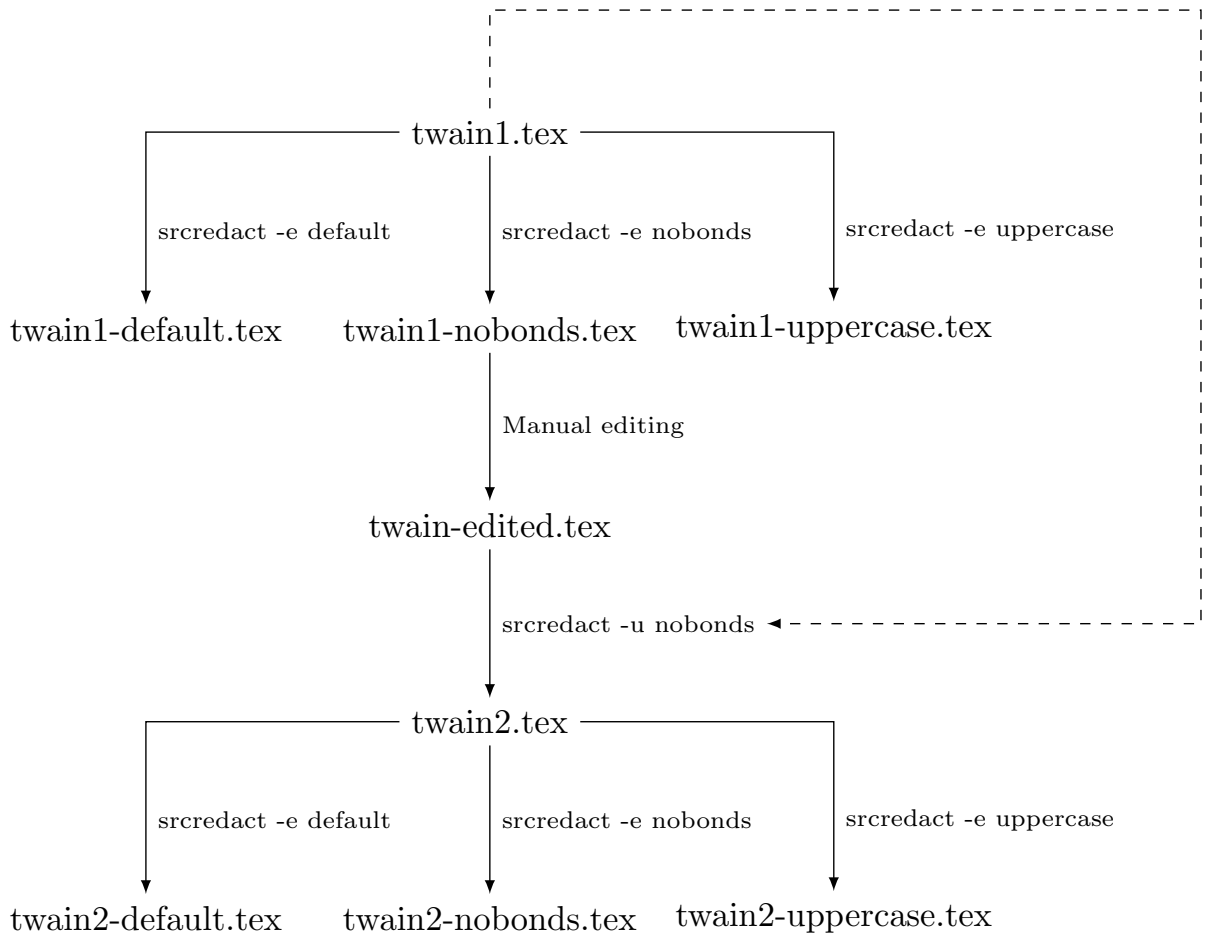


Figure 2: Document workflow with optional redactions

DocCenter — TEXing 11 million documents a year

Joachim Schrod

1 TEX goes banking

Our company's product DocCenter is used to manage the standardized, corporate identity (CI) conformant, written communication of a company. Our reference customer is 1822direkt,[1] a German online bank, which uses it to handle almost all written communication with its customers. Some communication is done via email or displayed as HTML messages in an online postbox, but most is created via LATEX and then printed or provided as PDF.

This experience report presents input methods, output formats, and delivery channels used by our customer. It also reports on related challenges and TEXnical solutions chosen.

But first things first: What is special about customer related documents of an online bank?

- They are small, mostly one or two pages.
- They are simple documents that don't need lots of markup, just some item lists, some fontifying, and some simple tables are sufficient.
- Few images are used, and those are not individually created for an individual document.
- All documents are obliged to use the company's corporate identity formatting rules; it should be hard for an author to break them.
- There are no reports, no books, no highly structured documents, no math — any use case where (LA)TEX is usually the first choice and what TEX experience reports are usually about is missing here.

So, small documents — but there are many of them; last year DocCenter was used to create 11 million of them! Quite a lot of the documents are letters:

- either with standardized content and a few fill-in variable parts
- or completely individual content
- or built from pre-defined text fragments

Even though the content itself is simple, it has to be enriched by corporate identity requirements, demands for standardized greetings and closings, automated addition of signatures of writer and department heads, footnotes with ever-changing advertisements for special opportunities, logos of current ratings from financial journals, etc. This enrichment is standardized and delivered by appropriate LATEX document classes; the content authors don't need to worry about it.

[1] 1822direkt is the online sales company of Frankfurter Sparkasse 1822.

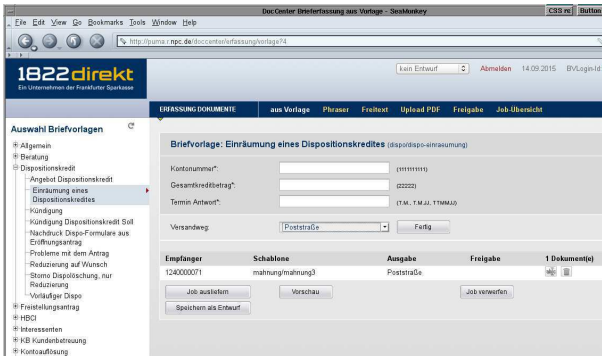


Figure 1: Standard letter input form (cropped).

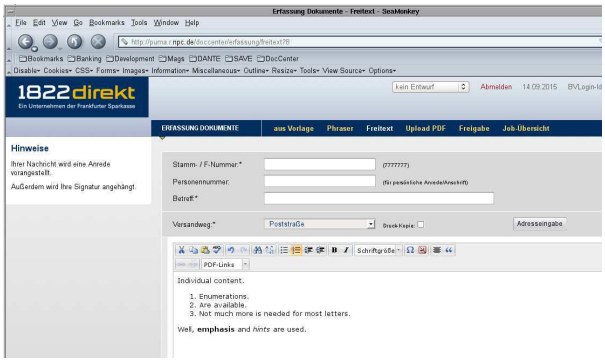


Figure 2: Individual letter input form (cropped).

In fact, the letter idiom is also used for account statements, credit card statements, and share notes. In fact, these are the majority of documents produced. These kinds of documents also often have special requirements concerning delivery, users may need to acknowledge the receipt, a printed version may be needed to be sent by post if no such acknowledgement arrives in due time.

Some other documents add additional requirements. A good example are PIN/TAN letters that must be kept private: they are printed on special paper only on certain in-house printers, they must be archived without the "secret part", etc.

2 DocCenter

As the introduction mentioned, we have two kinds of documents that are created with DocCenter: (1) letters that are created by staff members, and (2) automatically created documents.

2.1 Document creation by humans

DocCenter provides a web-based intranet application to create documents by staff members. Figures 1 and 2 show the basic interface.

Standardized letters are created by application forms that request only the variable parts. No letter content is shown during input of these *document parameters*, and it's not needed either: Staffers write those letters by the dozens or even hundreds; they know the letter's content by heart. Input must be quick and must be checked as closely as possible.

E.g., the author need not input a customer's name, or address, or account details — they are all available to be inserted into the letter content after the customer's account number has been input in the form. The actual letter content comes from so-called *templates*. We postpone describing their content creation to section 2.4, as the same templates are used for automatic document creation as well.

A preview is available to check final output — but most often it is not used. Users rely on proper document creation; they need to get the letters out in the most efficient way.

Individual letters may also be created, with boilerplate text parts available to ease that duty. For that task an HTML editor, TinyMCE, is integrated. That editor is configured to provide only formatting capabilities that are CI-conformant. The editor's XHTML result is then converted to LaTeX and output over the chosen channel.

Users of this "frontend" to DocCenter work on a few small documents at a time, and expect very fast system reaction time, both for preview and output creation. This speed aspect is called *latency*: Document generation and formatting on the server must happen fast, with results delivered very quickly.

2.2 Automatic document creation

DocCenter provides an HTTP interface to create jobs with documents. Most prominently, that interface is used for automated document mass production. Those documents are still small, but tens of thousands may be requested in one job.

Such a job's document requests don't include the content to be output. They name a document template which determines the content or how the content is to be generated. Document parameters in the request configure output or content creation. Other request attributes establish the output channel to be used, e.g., printer, online PDF delivery, transfer to a print shop, or others.

Since this interface is not used by humans, quick reaction time is not important. On the other hand, it is important that jobs are finished in some predetermined given time. E.g., when account or credit card statements are created for all customers once a month, there are service level agreements to fulfill — these statements must be delivered to the customer by a specified date.

This is a different kind of performance demand than the requirement of low latency for interactive usage: Commonly called *throughput*, it is concerned with overall processing time for a given set of documents, not about the processing time for a single document.

2.3 Basic architecture

DocCenter uses basically a 4-step lifecycle of document processing that provides a maximum of stability and control:

generate ⇒ format ⇒ output ⇒ archive

Document variants are used to provide needed specialization for one or several of those phases. The most important are variations of document generation. They are implemented as *plugin classes* that may be added to the system as needed. Thus, new demands for specific content generation or formatting can be satisfied easily by realizing and deploying a new plugin, without changing the base system.

Some illustrations:

Account statements use a specific generation step that fetches account data from the database; a document's LaTeX markup is created completely by the application.

Standardized letters generate a LaTeX document file that merely contains parameter declarations and then inputs a LaTeX template file that may use the parameters. (Those parameters are basically macro declarations.)

Individual letters transform XHTML content to LaTeX markup and content during generation; XSLT is used for that.

PIN letters don't archive the actual PIN, just the letter text. Thus bank personnel later can see when a PIN letter was sent to which person, but not the actual secret initial PIN data.

Other document variants provide further specializations that are even more company-specific. Realization of such document variants is the primary method for adapting DocCenter to different customers' demands.

2.4 LaTeX templates

Standardized letters are by far the most typical document that are created by DocCenter — not by document numbers, account statements dwarf that — but by variety that is handled smoothly with our application.

Within a bank, new content for a letter may not be easily created by a staff member on a personal

whim. Professional content (correct terms, factual correctness) is provided by one organizational unit, another checks that CI-style conformant phrases are used, while wording is checked by a third one. Each new letter and each change must go through the required protocols and have approvals recorded, to be available for inspection by auditors.

For that reason, actual document content is created and managed by an editorial team that organizes the process and coordinates the different groups that are involved. This editorial team actually creates LaTeX and XML files with meta-information; no interactive frontend is involved that hides that technology.

An advantage that we did not hesitate to exploit is the simple document structure. TeX, and subsequently LaTeX, assigns special functionality to many characters, be it `$` to introduce math mode, `%` to start a comment, or other characters used to make markup of complex documents more readable. Well, we don't have complex documents, we've got no math to typeset, and comments in documents are an alien concept that's hard to communicate to staff members anyhow. On the other hand, not being able to simply type `$` or `%` in a letter from a bank to get the respective characters in the output — that's a difficult restriction for this kind of user.

For these reasons, we reconfigured LaTeX a bit and added application-specific markup that supports creating simple documents in a way that can easily be learned by non-TeXies while still being LaTeX:

- Very simple LaTeX
- No math
- A minimum of special characters: just `\ { }`
- In particular, $ and % are normal characters, lest they create havoc in our banking context
- Insert document parameters, with optional formatting
- Optional text, controlled by parameters
- Some additional special environments; e.g., creating a pre-filled answer letter to the bank that is appended to the actual letter.

From the usual TeX point of view, these would hinder creating reports or longer documents. For our target use case, creating letters, such a reasonable subset of LaTeX functionality enables users to create new standard letter templates within hours, without a steep learning curve. (The few hours are actually spent by learning what metadata is needed and how to express it, not by creating real content.)

3 Challenges

Document creation in the context described above comes with some unusual challenges. Solutions are readily available in the TeX world, if we look beyond common knowledge of how a contemporary TeX system is used.

3.1 Output variations without reformatting

DocCenter has to be able to output formatted results via different *output channels*:

- PDF, to be delivered online to the customer
- Printed in-house, on both PostScript and PCL printers
- Print files for external print shop

Output is *not* the same for these output channels:

- Online PDF usually needs an embedded letterhead, provided as an image.
- Print output uses letterhead paper; thus a letterhead image must not be embedded.
- Documents use different types of paper; e.g., first page on letterhead paper, second page on white paper, maybe third page again on letterhead paper.
- Printer-specific tray control: Each printer may have paper types in different trays; using the right paper type as per requirement above must thus be configured and realized *per printer*.
- Printer calibration: Precise output positioning is important for letters, address fields must fit exactly into window envelopes.
  - Each laser printer feeds paper a bit differently, output doesn't end up on the page where it should be.
  - Experience shows that different printers may stray up to 5mm (0.2in) in all directions; while positioning errors for different sheets in one printer is a magnitude lower.
  - Therefore we need a *per printer configuration* (again) that offsets output on the page.
- Folding machine control: Output on some special printers is fed immediately to a folding machine that controls the completion of all of a letter's pages, folds them, and places them into an envelope. This is controlled by bar codes at the paper's left edge — these bar codes only have to be inserted when printed on these printers, not for any other output channel.
- Print shops need associated metadata (some want them embedded invisibly into PDF files) for paper type control.
- Some print shops always print duplex; extra empty pages may have to be inserted for them.

- Preview output needs watermarks (grey "draft" in the background), to make sure that the four-eyes control workflow cannot be easily circumvented.
- Output of archived documents again needs a watermark, to distinguish original documents sent to the customer from internally produced copies.
- Some letters are archived incompletely, e.g., secret PIN numbers must not be stored. One should still be able to have a partial view of the archived document, without that secret part.

But the biggest issue of all is the requirement that output of a document may have to be repeated on a different output channel after months or even years — taking the peculiar requirements above into account — *without reformatting the document*. Reformatting a document after several years always has the risk that the output may be different, owing to changed LaTeX packages or other internal macro changes. That must not happen; precisely the same document has to be reproduced.

Therefore, to separate format from output phase in DocCenter's document life cycle doesn't just mean that a formatted result, e.g., a PDF file, is sent to some output device. Instead, the output phase transforms the formatting result and implements the output channel specific requirements.

The current standard in TeX world for output format is PDF. Almost all current publications and presentations at conferences take that for granted. While we could realize all this output phase manipulation by transforming TeX-produced PDF files, the overall TeX universe has an older technology available that's better suited for our purpose:

DVI files with `\specials`

TeX specials are used in the DVI format result to declare the need for duplex/simplex, letterheads, paper types, watermarks, etc. DVI drivers interpret them and produce adequate printer-specific output.

As an example, printer-specific tray control is as easy as setting up directories with include files with standardized names; these include files contain printer control commands to access trays for the correct paper type. Inclusion of these files is triggered by appropriate `\special`s in the document.

Printer-specific output placement is even easier to realize: Every DVI driver has options for offset control, a configuration file per printer has values for that option.

Last, but not least, using DVI greatly lowers cost for our document archive. A typical letter with roughly 40 KB in PDF format needs only 2 KB in DVI format. The disk space requirement is thus reduced from 500 GB per year to 25 GB per year. This doesn't sound much in terms of today's USB storage prices where you get multiple TBs for cheap — but you can't use such inexpensive storage easily in a bank's data center. There it still matters if a 10-year storage archive needs 10 TB or 1 TB.

3.2 Latency improvement

Before DocCenter was deployed, a predecessor system was used that was also based on LaTeX. With a rather naive implementation, that needed 1.5 seconds to process a document. Adding the communication latency, delivering a preview from server to user needed up to 3–4 seconds, clearly far too long.

Root cause analysis showed us the reasons for that behavior: Most of the time was spent in boilerplate processing: reading and processing LaTeX class and package files, font configurations, etc. Creating a letter with two paragraphs of text needed processing more than a dozen macro files. Actual time for formatting the document's content was minimal. (SSD disk caches might have helped, but were not readily available in the clustered server architecture that is in use at the customer.) Additional time was spent by processing each document twice, as is common document production practice in the TeX world.

Well, that problem was easily tackled with standard TeX techniques from the early ages: We don't have document-specific packages, and our documents are not one-off creations. Instead, all of our 11 million documents use the same set of packages, maybe with some small variation in feature usage. So we created a TeX format file that has LaTeX and all used packages and font definitions preloaded. The format also redefines `\documentclass` and other preamble control sequences to do nothing — class and package files are already loaded, after all.

A second measure was to stop processing documents twice. Analysis showed that we don't need any of LaTeX's features, like cross references, that demand multiple formatting runs. Processing each document once is sufficient.

Reading a format file is *very* fast in TeX, being the equivalent of a memory dump. Document formatting time was reduced from 1.5 seconds per documents to 0.06 seconds per document; a 25 x improvement by using our specific FMT file and doing only one run.

3.3 Throughput improvement

Creating account statements was another challenge. The predecessor system used a tabular layout, as is common with such statements, implemented via

LaTeX's longtable package. Each of the hundreds of thousands of statements that had to be produced was created separately, with a new database connection and queries, creation of a new LaTeX file, running LaTeX twice (owing to usage of longtable), and creation of output files for the customer.

To achieve a service level of maximum processing time of 24 hours, the creation process was spread over 10 systems where it needed a total time of 22 hours. At least half of that time could be attributed to non-optimal usage of TeX technology.

Our first observation was that "looks like a table" doesn't mean that it is a longtable or even a tabular environment in LaTeX markup parlance. Bank statement layout is not at all flexible; column widths are preset and don't change with content. There are some running heads at page breaks, but they don't demand the full power of complex table formatting capabilities.

Instead, we turn towards the most basic formatting capability TeX has: `\hbox` and `\vbox`. Nothing is faster in TeX formatting than using these primitives. A booking entry in the statement is not a table line with columns, it's an `\hbox` that contains `\vbox`es with fixed widths. Voilà, blindingly fast processing by TeX is the result.

Our second observation was, again, the overhead of boilerplate processing for all these document files, as mentioned in the previous section. For this use case, we optimized it even further, beyond using our own FMT file. We generate markup and content for ca. 50,000 documents in one run and feed them directly to LaTeX, without any intervening process. LaTeX then dutifully produces a DVI file with 200,000–250,000 pages.

The choice of DVI files to represent our formatted result comes in quite handy now. In a DVI file, pages are linked from the back to the start. The first 10 TeX counters are stored at such a page start. Our macros store a document ID in one of these counters and so we can detect the start of a new document while jumping from one page to the next. Splitting the single DVI file into 50,000 smaller ones is thus a matter of less than a second, mostly dominated by I/O times on the networked storage in use in such clustered environments.

An interesting point from a TeX point of view is a further optimization that made splitting much easier. A hairy detail of DVI file splitting is the declaration of fonts: they appear at first use and at the end of the file. Rather complicated logic is needed for an arbitrary split algorithm to handle that properly. However, since we are in such a restricted use case scenario, we can add a first page that does nothing but load all fonts needed. Our DVI-splitting algorithm collects all font definitions from that first page, to be output to every produced DVI file at the start, and then may append DVI pages from the respective document without having to worry about appearance of font definitions at all. This makes the split code size really small, robust, and easy to maintain — much easier than comparable code for splitting a large PDF file.

Using this production approach has proved a full success: We achieved a performance improvement of a factor of 100. All statements can be created on one system within 2 hours.

4 Conclusion

Using TeX as the centerpiece for document creation in DocCenter was a full success. We have a robust application that's purring on without production problems in that area. Traditional TeX toolbox solutions like DVI files, specials, FMT files and the like are still immensely useful for the diversified requirements of document generation, even in today's communication demands. It will still work in 10 years, of great importance for a bank. What other typesetting system can say the same?

Still, there were some minor hurdles that we had to overcome:

- A standardized and stable API for TeX processing is missing. While it's accepted that TeX is hard to handle by humans, it's also hard to control by an application.
- Most important, batch mode and error message handling are not perfect for monitoring.
- On the organizational side, there is no staff easily available with sufficient knowledge of (La)TeX technology — personnel for support tasks is even harder to find than for development.
- Customer-specified special formatting for individual documents and one-off-changes are difficult to achieve on-site by the customer's editorial group, without our involvement.
- Quality of DVI drivers is worse than 15 years ago. (`dvips` is the exception.)

But these issues shouldn't stop you from using TeX for similar tasks. Other typesetting systems will come with their own problems, and more of it — we have the scars to prove it, but that's another story.

◇ Joachim Schrod
Net & Publication Consultance GmbH
jschrod (at) acm dot org

Preparing LaTeX classes for journal articles and university theses

Tom Hejda

Abstract

There is a substantial difference between the requirements on a LaTeX class for a scientific journal and for university theses. The main point is that a journal class is by definition *restrictive* — the journal has to be very keen on the precise look and structure of the articles, whereas the thesis class is by definition *modular* — different theses ask for a slightly different layout and structure, some have appendices and some do not, etc. We discuss the differences and their implications on the class design.

1 Introduction

It is natural that different types of documents ask for different LaTeX classes. We will discuss the differences for journal articles and university theses. This is partly a response to a recent boom in LaTeX classes for theses issued and enforced by universities, where it is commonly seen that the classes do not meet good standards, students have difficulties using them and the result is in many cases far from satisfactory. Even though this is the case, we refrain from giving bad examples, and we rather focus on the core ideas that should be behind the design of such a class.

This paper is organized very simply. In the next two sections, we discuss the demands on classes with different purposes. In Section 4, we describe the solution to the demands that were used to design and code the `ctuthesis` class that is being developed at the Czech Technical University (CTU) in Prague. We believe that our proposed solution serves as a good example of how things *can* be done.

It should be noted that while graphical design plays an important role in the publication process, we will omit the discussion about graphics as this is mostly irrelevant to our points. We merely note that the class `ctuthesis` that will be used as an example is based on a plain TeX class called `ctustyle` [3], to which the next article in this *TUGboat* issue is devoted.

2 Different documents are made differently

The typical workflow for publishing articles in scientific journals involves several steps:

1. **Primary submission by the authors** — it need not be in the journal's style and need not strictly follow the typographical policies of the journal.
2. **A referee process leading to an accepted version** (or rejection, but that is not interesting for us) — at the end of this process, the authors provide a version of the article that should comply with all the in-house policies.
3. **In-house typesetting and editorial copy preparation** — The staff of the publisher take the sources (code, figures, etc.) and prepare the article to their liking.
4. **Proofreading** — the authors point out any mistakes made during the typesetting and possibly other things they do not like.

If we look into how theses are usually typeset, we see that most often the last two steps are missing: There is no one to typeset the thesis in a professional way nor to control the way that the thesis is typeset. This means that the thesis author is in some sense much more responsible for his work than the author of an article, at least from the typographical point of view.

3 Variety of documents

A second big difference between the two class types is in the variety of documents. In general, most articles in a single journal follow a similar scheme for sectioning, floating objects, references etc.; also, they are usually from a rather narrow field.

On the other hand, a single university has many faculties with many branches of study, and it is clear that a programming thesis looks significantly different from a theological text or an architectural study. It is quite natural that the first one will contain a lot of code samples and probably a reference manual, the second one will be basically a long text with a lot of direct quotes of paragraphs from other sources, and the third will contain a long graphical appendix. Also, the thesis is the student's child and he should be able to make it look as he likes, within the requirements.

To allow a single LaTeX class to accommodate all these needs, the class has to be highly modular; the presence of appendices has to be optional, for instance, and in general, more or less everything has to be configurable. The class should have only minimal fixed design in order to comply with the requirements of the university.

4 Our solution

The solution for article classes used by `actapoly` (the journal of the CTU) [1] does not involve any special tools — article authors set up the metadata of the article and these are then used by `\maketitle` to print the article title block. All the standard LaTeX

environments and commands such as sectioning commands, lists, floats, tables etc. are then given a fixed graphical design that forms the graphical identity of the journal. This is what nearly every journal publisher does in their class files.

By comparison, in designing our class — called `ctuthesis` [2] — for university theses, we needed a high level of modularity, as discussed. This is allowed mostly by two important ingredients:

1. **Good key-value interface.** Most modifiers of the class behaviour are implemented using this interface. The interface itself is coded using the very usable and highly versatile `l3keys` package. In general, the whole class is written in `expl3` as much as possible.
2. **Two-phase class and package loading.** The idea can be seen in Figure 1 — we load the class, then set everything up using the key-value interface, and then the command `\ctuprocess` inputs another file of the class. This additional file contains a lot of conditional package loading and package setup.

There are several types of keys for the key-value interface:

- appearance keys — languages, colours, a switch for the inclusion of the list of figures, etc.;
- metadata keys — title, subtitle, author, supervisor, name of the department, and a lot of other information;
- package options — customizable loading of certain packages for which it makes sense, including for instance: `amsthm` (since someone may prefer `ntheorem` or another package and there is no reason to forbid it), `listings` (we set up the listings design in a particular way that someone may not like), or `hyperref` (since it is sensitive to the order in which the packages are loaded and making it conditional can help in resolving the issue).

Also, in the internal design, we borrow the idea that is seen in `beamer` — namely what we call *templates* and *fields*. Examples are worth complicated explanations, so as an example, the titlepage, or the list of figures in the two-column frontmatter make up typical templates, whereas fields are things such as the title and the abstract (these are actually language-dependent, so we have a field for the title in all languages in which it is needed, and similarly for the abstract, the university name, etc.), the name of the author, the address of the supervisor, etc.

Also, there is an interface for *themes* — it could happen that a faculty of the university had a special requirement that "supervisor" should be called

```
\documentclass{ctuthesis}
\ctusetup{
   key1 = value1,
   key2 = value2,
   ...
}
\ctuprocess

% ... user stuff goes here ...

\begin{document}
\maketitle
...
```

Figure 1: Structure of the preamble of a document in `ctuthesis`.

"project manager", and this is possible using a theme for this faculty that changes `\supervisorname` in the `english` language. It is of course possible to implement this without the themes interface, but it would mean adding strange conditionals at strange places in the class files for one-off issues like this one. We do, however, store all templates and themes in a single file with a clear structure.

5 Concluding remarks

To conclude, let us mention the most important points of the paper:

1. Different document types need different class designs.
2. Class authors should think of how the class will be used and who the users will be.
3. The more the users will interact with the class, the cleaner the class interface should be.

References

[1] Czech Technical University in Prague. Acta Polytechnica — submissions. `https://ojs.cvut.cz/ojs/index.php/ap/about/submissions#authorGuidelines` [2015-08-01].

[2] Tom Hejda. LaTeX template for theses at CTU in Prague. `https://github.com/tohecz/ctuthesis` [2015-08-01].

[3] Petr Olšák. CTUstyle — Plain TeX template for theses at CTU in Prague. `http://petr.olsak.net/ctustyle-e.html` [2015-08-01].

◇ Tom Hejda
Dept. Math. FNSPE, Czech Technical University in Prague
Trojanova 13
Prague 12000
Czechia
`tohecz (at) gmail dot com`
`http://github.com/tohecz/`

The CTUstyle template for student theses

Petr Olšák

Tom Hejda introduced his work on a template for student theses at the TUG 2015 conference. The template is used at the Czech Technical University in Prague (CTU). The present article is intended as a companion to Tom's article "Preparing LaTeX classes for journal articles and university theses". In his article, the LaTeX point of view and comparison with another project is highlighted. In this article, on the other hand, the original development of the template (which has nothing to do with LaTeX) and the grounds for some typographical decisions are mentioned.

Beginnings

In October 2012, students of the Czech Technical University in Prague started a discussion in an Internet forum [1] about the need for a good template for Bachelor, Master and Doctoral theses at our university. They mentioned that other universities have an interesting LaTeX template but CTU uses nothing centrally, while there are various LaTeX solutions with not-so-good typographical design in a few departments.

I gave my little contribution to this discussion forum too. I announced that I am able to suggest typographical design and I can implement this template by plain TeX macros only. I never offer a LaTeX solution because LaTeX is a bad way of TeX usage from my point of view and I don't want to offer something that I believe is bad. Then I waited many months to see if somebody else would offer a LaTeX solution, but this did not happen. So, I started with typographical design and the implementation of student theses in January 2013. The implementation was based on my plain TeX OPmac macros [2] and the template was named CTUstyle [3].

Several students started to use my CTUstyle template and many of them complimented me that the template has a good design and it is simple to use. I want to emphasize that many of these students had no previous knowledge about TeX nor LaTeX, but they were able to simply use this template. This is contrary to the opinion that LaTeX is simpler to apply than plain TeX at the user level. This is not true when a good template is available.

Meanwhile, the template designer (like me) can realize a typographical design much more straightforwardly in plain TeX, because only primitives are used (like `\hbox` and `\vbox`) and the designer's time isn't dissipated by useless complexity in LaTeX.

Principles of thesis templates

We can consider the thesis template from different and independent points of view:

- The typographical design (independent of the software used).
- The implementation of university rules (for example what information must be in the first pages and where).
- The user interface.

The typographical issues will be mentioned in the next section of this article. The other two points of view can be illustrated by the following minimal example of CTUstyle usage:

```
\input ctustyle
\worktype [M/EN]  % Master's thesis, in English
\faculty    {F3}  % one of 8 faculties at CTU
\department {Department of special studies}
\title      {Minimal Document}
\author     {Ben A Uthor}
\date       {January 2013}
\abstractEN
    {This document is for testing purposes only.}
\abstractCZ
    {Tento dokument je pouze pro test.}
\declaration
    {I hereby declare I didn't monkey around.}
\makefront

\chap Introduction

The introductory text.

\sec The Idea

My big idea for this Master thesis is...

\bye
```

The user environment is designed to be like "filling in a form". There are several mandatory fields which must be set. All of them are used in the minimal example above: `\worktype`, `\faculty`, `\author` etc. If a mandatory field is omitted then an appropriate error message is printed. The mandatory fields declare a minimum of information needed by university rules.

Optional fields can be defined here as well; for example, `\subtitle`, `\supervisor`, `\authorinfo`, `\thanks`, etc. All fields are documented in the CTUstyle documentation, which was created with the CTUstyle template in order to show the usage and the design of the template.

Then the `\makefront` command generates the first automatically created pages: title page, the declaration and thanks page, the abstract page (in two languages), the table of contents, the list of figures or tables (if present). For example, the full

Fig. 1. The CTUstyle title page.
This thesis by G. Šourek is fully available at [9].

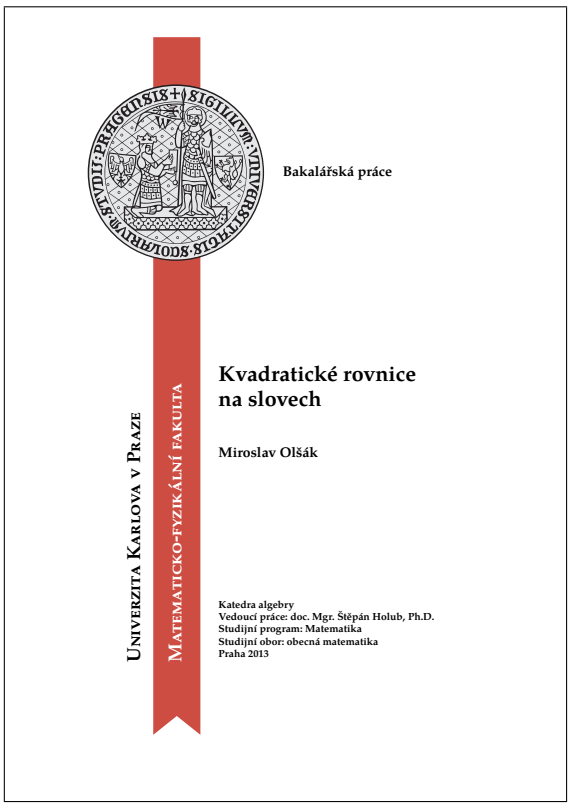


Fig. 2. The CUstyle title page
This thesis by M. Olšák is fully available at [10].

name of the faculty is printed in the title page but user needn't specify it; e.g., F3 means Faculty of Electrical Engineering.

The body of the work is structured using OPmac macros like `\chap` for chapter, `\sec` for section etc. These commands are described in OPmac documentation in detail and roughly in CTUstyle documentation.

The typographical design

There are two big universities in Prague: Charles University (CU) [6] and Czech Technical University (CTU) [7]. I did the CTUstyle [3] template first for CTU and afterwards, I modified this template to CUstyle [4] for CU.

The CTU is a technical university founded 300 years ago. I intended to create the template for CTU with a modern look and feel which can be used by students with enjoyment. I chose the technical font Latin Modern for the template because CTU is a technical university. I used the color decreed by the corporate identity of this university: blue Pantone 300 C. The complementary color (orange) is used as navigation color only for marking active hyperlinks and only in online version (disabled during printing).

I decided to use the heavy blue rule seen in Fig. 1, inspired by the typographical manual of CTU. The first automatically generated pages (except the title page) are designed as two column. The body of the work is one-column and the pages are numbered from one in the body. This is one of the university's rules.

The whole work is designed as a two-side book with running heads that disappear into the middle of the binding. I assume that the document will be printed using a duplex printer with the capability of printing color. A non-duplex variant of the document can be set by `\onesideprinting` and a grayscale variant by `\blackwhite` but it is not recommended.

The CUstyle for the Charles University copies the main principles of the design from CTUstyle, but the Pagella font was chosen because it looks more ancient. CU was founded in the Middle Ages in 1348 by Charles IV, King of Bohemia and King of the Romans. It was the first university in the Middle Europe. So the design of the font and of the title page reflects the history of CU. The red rule on the title page means the ribbon with the seal used by kings. On the other hand, the red color is specified in the corporate identity of this university.

Fig. 3. The CTUstyle TOC page
This thesis by G. Šourek is fully available at [9].

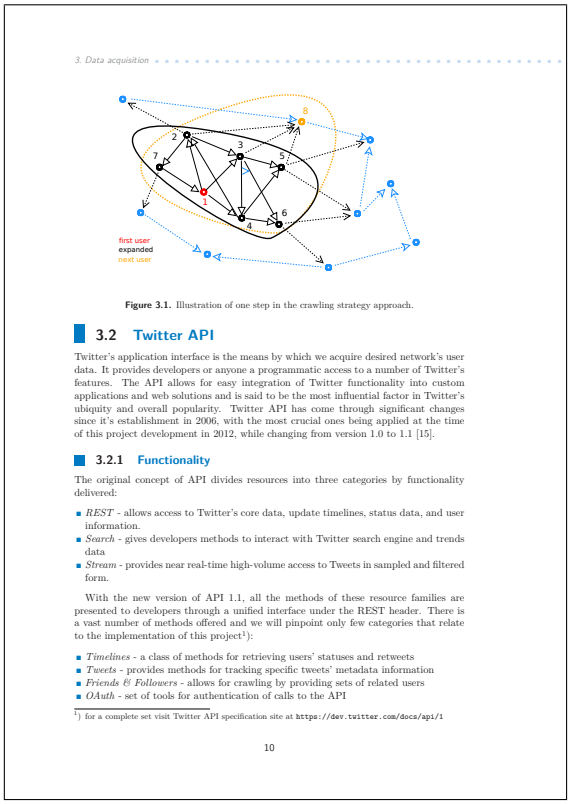


Fig. 4. The CTUstyle common page
This thesis by G. Šourek is fully available at [9].

Figures 1 to 4 show selected pages from theses using CTUstyle and CUstyle.

Next developments

I offer my help to students who are using my plain TEX templates. I make occasional small improvements to the template due to requests from the students. There are no big demands. I am happy that I have received many compliments and thanks from the students.

For example, I've recently created a new template for plain TEX slides with the same look as CTUstyle called CTUslides [5]. Students can do their presentations in the same style.

Sometimes, a suggestion is made at the discussion forum [1] like: "I wish to use this template but LATEX seems better for me". I replied: "The typographical design is done and it is independent of the software used. You can use it in MS Word, for example, or in LATEX. I don't recommend either but people can prefer something different than I. Anybody can do the implementation of my template with other software." But no such "anybody" appeared after two years. So, I decided to contact Tom Hejda, because I knew him as `tohecz`, a user from `tex.sx` [8]. And he accepted the proposal (with financial support by Prof. Hlaváč from CTU). This is another story, described in the previous article ...

References

1. `http://forum.fel.cvut.cz/topic/3697/`
2. `http://petr.olsak.net/opmac-e.html`
3. `http://petr.olsak.net/ctustyle-e.html`
4. `http://petr.olsak.net/custyle-e.html`
5. `http://petr.olsak.net/ftp/olsak/ctustyle/slides.pdf`
6. `http://www.cuni.cz/UKEN-1.html`
7. `http://www.cvut.cz/`
8. `http://tex.stackexchange.com/`
9. `http://cyberold.felk.cvut.cz/research/theses/papers/339.pdf`
10. `http://www.olsak.net/mirek/bakalarka/`

◇ Petr Olšák
Czech Technical University
in Prague
`http://petr.olsak.net`

New multibibliography package *nmbib*

Boris Veytsman and Michael Cohen

Abstract

Two years ago we presented a *multibibliography* package that provides multiple lists of citations with alternate orderings. The *nmbib* package is a complete refactoring of that program. It offers a broader variety of citation commands, streamlines the creation of bibliographies, ensures compatibility with the *natbib* package, and provides other improvements.

1 Introduction

In scientific books and papers, the bibliography is traditionally relegated to "back matter". One might think of it as less important than the main text, being added almost as an afterthought. However, this is not correct. Actually the bibliography tells an important story about the field of study and the place of the given work within it.

Correspondingly, different orderings of the bibliography list tell different stories. A sequential ordering shows the logic of the current work, while an alphabetical ordering shows the contributions of different people to the field. Another possibility, a chronological ordering, is much less common. It shows the history of the field, and thus can also be important and interesting.

Which story should authors choose for their work? We argue that they may have all of them, providing several differently ordered lists instead of just one. Indeed, computers make creation and shuffling of bibliographies very simple, and electronic publishing eliminates the problem of dead trees needed to print additional pages (Cohen, 2014).

Accordingly, some time ago we proposed a way to produce several differently ordered bibliographies (Cohen et al., 2013a), and released the package *multibibliography* (Cohen et al., 2013b), implementing it for LaTeX and BibTeX. This package was a proof of concept for the ideas of the work [2]. It allowed creation of three bibliography lists with different styles: alphabetical by authors, chronological by date, and ordered by the appearance in the text. Each `\cite` command produced entries for all three lists. There were clickable links among the lists and from articulated citations to the lists.

This prototype implementation, while showing the usefulness of the idea, had a number of limitations. First, the only format of the citation allowed was the following: "[Cohen et al., 2013a: 2]". A user wanting purely author-year or numeric style would be frustrated. Second, the BibTeX styles for the reference lists were fixed. If a journal or a book required different punctuation or capitalization than that provided, there was no way to adjust the format of the references. Third, the Perl script used to manage the lists was a rather *ad hoc* solution and needed some refactoring.

In this paper we describe the new package *nmbib* (Veytsman and Cohen, 2015), including many new features.

2 Features of the *nmbib* package

Like our previous package *multibibliography* [3], the *nmbib* package can create three bibliography lists: timeline, sequential, and alphabetical. However, unlike *multibibliography*, it allows the user to easily omit any of these lists.

The main feature of *nmbib* is full compatibility with the famous *natbib* package by Daly (2010). (In fact, `nmbib.sty` loads `natbib.sty`.) Compatibility means two things. First, any citation command of *natbib*, such as `\citet`, `\citep`, `\citeauthor`, `\citeyear`, etc., works directly. Second, any *natbib* `bst` style can be used for the respective bibliography. This includes customized styles created with the *makebst* package (Daly, 2003). Thus a user can easily create a bibliography style according to any specification.

The package is also compatible with *hyperref* (Rahtz and Oberdiek, 2012). If the latter is loaded, all citations have hyperlinks with evident properties: clicking on authors' names lands the user on the alphabetical list, clicking on a date lands her on the chronological list, and clicking on a number goes to the sequential list. There are hyperlinks among the labels of the bibliography items with the same meaning. Moreover, if one uses the styles supplied with the package, there are additional links among the *bodies* of the bibitems: for example, clicking on authors' names in the chronological list will bring the user to the relevant entry in the alphabetical list.

The package can be extended to new types of sorting. For example, if the bibliography entries have a field with the number of citations for the given paper, we can imagine a list ordered according to the influence of the publications.[1]

3 User interface

The full manual [4] for the package is available on CTAN. Here we discuss just the main features of the program.

To produce a standard BibTeX-based bibliography, one uses three types of commands:

[1] We are grateful for this suggestion to the audience of TUG 2015.

1. Citation command: `\cite` (and, for *natbib*, a number of extensions such as `\citeauthor`, `\citenum`, etc.).
2. Command for setting up the bibliography style: `\bibliographystyle`.
3. Command for setting the bibliography databases and printing the bibliography: `\bibliography`.

Our interface is designed following the same pattern. First, it uses the same `\cite` commands. Our package allows one to intermix author-year and numerical citations, as in this paper. The command `\citefull` may be used to get the full citation in the *multibibliography* package style.

Next, our command `\multibibliographystyle` is similar to the command `\bibliographystyle`, with the following important difference: the user must separately set styles for the three major kinds of bibliography: `timeline`, `sequence`, and `authors`. For example:

```
\multibibliographystyle{timeline}%
      {chronoplainnm}
\multibibliographystyle{sequence}{unsrtnm}
\multibibliographystyle{authors}{plainnm}
```

The three BibTeX styles referenced here are supplied with the package. As discussed above, one can use any *natbib*-compatible style for alphabetical and sequential lists.

Finally, the command `\bibliography` in standard BibTeX use has two functions: setting the databases *and* also printing the bibliography. Our package separates these functions: the command `\multibibliography` only sets the databases, while the list of references is printed with the command `\printbibliography`. The latter has one argument, which sets the type of the list, as in

```
\printbibliography{sequence}
\printbibliography{authors}
\printbibliography{timeline}
```

In the standard BibTeX-based workflow, after a `latex` run, a file `\jobname.aux` is processed by the `bibtex` program, creating the file `\jobname.bbl`. In the *nmbib* workflow, three files are generated: `\jobname-timeline.aux`, `\jobname-sequence.aux`, and `\jobname-author.aux`. Each of them should be processed with `bibtex`. The script `nmbibtex`, supplied with the package *nmbib*, can automate this processing, but is not required.

The package *nmbib* is highly customizable: it is easy to change bibliography labels, names of the individual lists, and other features. See the manual for a full description.

4 Conclusions

We have developed and released a completely new implementation of the *multibibliography* package. The new program has a flexible and highly customizable interface.

Sequential bibliography

[1: Cohen (2014)] Michael Cohen. From Killing Trees to Executing Bits: A Survey of Computer-Enabled Reading Enhancements for Evolving Literacy. In *VSMM: Proc. Int. Conf. on Virtual Systems and Multimedia*, Hong Kong, December 2014. http://www.vsmm2014.org, ISBN 978-1-4799-7227-2, https://www.researchgate.net/publication/277006068_From_Killing_Trees_to_Executing_Bits_A_Survey_of_Computer-Enabled_Reading_Enhancements_for_Evolving_Literacy.

[2: Cohen et al. (2013a)] Michael Cohen, Yannis Haralambous, and Boris Veytsman. The multibibliography package. *TUGboat*, 34(3):340–343, 2013. http://tug.org/TUGboat/tb34-3/tb108cohen.pdf.

[3: Cohen et al. (2013b)] Michael Cohen, Yannis Haralambous, and Boris Veytsman. *The Multibibliography package*, March 2013. http://ctan.org/pkg/multibibliography.

[4: Veytsman and Cohen (2015)] Boris Veytsman and Michael Cohen. *New Multibibliography Package nmbib*, July 2015. http://ctan.org/pkg/nmbib.

[5: Daly (2010)] Patrick W. Daly. *Natural Sciences Citations and References (Author-Year and Numerical Schemes)*, September 2010. http://ctan.org/pkg/natbib.

[6: Daly (2003)] Patrick W. Daly. *Customizing Bibliographic Style Files*, September 2003. http://ctan.org/pkg/custom-bib.

[7: Rahtz and Oberdiek (2012)] Sebastian Rahtz and Heiko Oberdiek. *Hypertext Marks in LaTeX: A Manual for Hyperref*, November 2012. http://ctan.org/pkg/hyperref.

Alphabetic bibliography

[Cohen (2014); 1] Michael Cohen. From Killing Trees to Executing Bits: A Survey of Computer-Enabled Reading Enhancements for Evolving Literacy. In *VSMM: Proc. Int. Conf. on Virtual Systems and Multimedia*, Hong Kong, December 2014. http://www.vsmm2014.org, ISBN 978-1-4799-7227-2, https://www.researchgate.net/publication/277006068_From_Killing_Trees_to_Executing_Bits_A_Survey_of_Computer-Enabled_Reading_Enhancements_for_Evolving_Literacy.

[Cohen et al. (2013b); 3] Michael Cohen, Yannis Haralambous, and Boris Veytsman. *The Multibibliography package*, March 2013. http://ctan.org/pkg/multibibliography.

[Cohen et al. (2013a); 2] Michael Cohen, Yannis Haralambous, and Boris Veytsman. The multibibliography package. *TUGboat*, 34(3):340–343, 2013. http://tug.org/TUGboat/tb34-3/tb108cohen.pdf.

[Daly (2003); 6] Patrick W. Daly. *Customizing Bibliographic Style Files*, September 2003. http://ctan.org/pkg/custom-bib.

[Daly (2010); 5] Patrick W. Daly. *Natural Sciences Citations and References (Author-Year and Numerical Schemes)*, September 2010. http://ctan.org/pkg/natbib.

[Rahtz and Oberdiek (2012); 7] Sebastian Rahtz and Heiko Oberdiek. *Hypertext Marks in LaTeX: A Manual for Hyperref*, November 2012. http://ctan.org/pkg/hyperref.

[Veytsman and Cohen (2015); 4] Boris Veytsman and Michael Cohen. *New Multibibliography Package nmbib*, July 2015. http://ctan.org/pkg/nmbib.

Chronological bibliography

[2003: Daly; 6] Patrick W. Daly. *Customizing Bibliographic Style Files*, September 2003. http://ctan.org/pkg/custom-bib.

[2010: Daly; 5] Patrick W. Daly. *Natural Sciences Citations and References (Author-Year and Numerical Schemes)*, September 2010. http://ctan.org/pkg/natbib.

[2012: Rahtz and Oberdiek; 7] Sebastian Rahtz and Heiko Oberdiek. *Hypertext Marks in LaTeX: A Manual for Hyperref*, November 2012. http://ctan.org/pkg/hyperref.

[2013a: Cohen et al.; 2] Michael Cohen, Yannis Haralambous, and Boris Veytsman. The multibibliography package. *TUGboat*, 34(3):340–343, 2013. http://tug.org/TUGboat/tb34-3/tb108cohen.pdf.

[2013b: Cohen et al.; 3] Michael Cohen, Yannis Haralambous, and Boris Veytsman. *The Multibibliography package*, March 2013. http://ctan.org/pkg/multibibliography.

[2014: Cohen; 1] Michael Cohen. From Killing Trees to Executing Bits: A Survey of Computer-Enabled Reading Enhancements for Evolving Literacy. In *VSMM: Proc. Int. Conf. on Virtual Systems and Multimedia*, Hong Kong, December 2014. http://www.vsmm2014.org, ISBN 978-1-4799-7227-2, https://www.researchgate.net/publication/277006068_From_Killing_Trees_to_Executing_Bits_A_Survey_of_Computer-Enabled_Reading_Enhancements_for_Evolving_Literacy.

[2015: Veytsman and Cohen; 4] Boris Veytsman and Michael Cohen. *New Multibibliography Package nmbib*, July 2015. http://ctan.org/pkg/nmbib.

◇ Boris Veytsman
Systems Biology School and Computational Materials Science Center
MS 6A2
George Mason University
Fairfax, VA 22030
USA
borisv (at) lk dot net
borisv.lk.net

◇ Michael Cohen
Spatial Media Group
Computer Arts Lab.
University of Aizu
Aizu-Wakamatsu, Fukushima 965-8580
Japan
mcohen (at) u-aizu dot ac dot jp
www.u-aizu.ac.jp/~mcohen

Generating PDF/X- and PDF/A-compliant PDFs with pdfTeX — pdfx.sty

C. V. Radhakrishnan, Hàn Thế Thành,
Ross Moore and Peter Selinger

1 Introduction

The pdfx package (pdfx.sty) currently supports generation of PDF/X- and PDF/A-compliant documents using pdfTeX, in some variants of these standards. Support for additional standards, such as PDF/E and PDF/VT is also available; the complete list is in Section 2.1 below. By 'supports', we mean that the package provides correct and sufficient means to declare that a document conforms with a stated PDF variant (PDF/X, PDF/A, PDF/E, PDF/VT, etc.) along with the version and/or level of conformance. The package also allows appropriate metadata and color profile to be specified, according to the requirements of the PDF variant.

Metadata elements, most of which must ultimately be written as XML using the UTF-8 encoding, is provided via a file named \jobname.xmpdata, for the running LaTeX job. Without such a file, providing some required information as well as a large range of optional data, a fully validating PDF file cannot be achieved. The PDF can be created, having the correct visual appearance on all pages, but it will not pass validation checks. Section 2.2 describes how this file should be constructed.

What this package *does not* do is check for all the details of document structure and type of content that may be required (or restricted) within a PDF variant. For example, PDF/VT [11] requires well-structured parts, using Form XObject sections tagged as '/DPart'. Similarly PDF/A-1a (and 2a and 3a) [3–5] require a fully 'Tagged PDF', including a detailed structure tagging which envelops the complete contents of the document. This is beyond the current version of pdfTeX, as commonly shipped. So while this package provides enough to meet the declaration, metadata and font-handling aspects for these PDF/A variants, it is not sufficient to produce fully conforming PDFs. However, with extra pdfTeX-based software that *is* capable of producing 'Tagged PDF', this package can be used as part of the overall workflow to produce fully conforming documents.

1.1 PDF standards

PDF/X and PDF/A are umbrella terms used to denote several ISO standards [3–5, 12–14, 16, 17] that define different subsets of the PDF standard [1, 6]. The objective of PDF/X is to facilitate graphics exchange between document creator and printer and therefore has all requirements related to printing. For instance, in PDF/X, all fonts need to be embedded and all images need to be CMYK or spot colors. PDF/X-2 and PDF/X-3 accept calibrated RGB and CIELAB colors along with all other specifications of PDF/X. Since 2005 other variants of PDF/X have emerged, as extra effects (such as layering and transparency) have been supported within the PDF standard itself. The full range of versions and conformance supported in this package is discussed below in Section 2.1.

PDF/A defines a profile for archiving PDF documents, which ensures the documents can be reproduced in the exact same way in years to come. A key element to achieving this is that PDF/A documents are 100% self-contained. All the information needed to display the document in the same manner every time is embedded in the file. A PDF/A document is not permitted to be reliant on information from external sources. Other restrictions include avoidance of audio/video content, JavaScript and encryption. Mandatory inclusion of fonts, color profile and standards-based metadata are absolutely essential for PDF/A. Later versions allow for use of image compression and file attachments.

PDF/E is an ISO standard [8] intended for documents used in engineering workflows. PDF/VT [11] allows for high-volume customised form printing, such as utility bills. PDF/UA ('Universal Accessibility') is emerging as a standard [9, 10] supporting Assistive Technologies, incorporating web-accessibility guidelines (WCAG) for electronic documents. In future, PDF/H may emerge, for health records and medical-related documents. Other applications can be envisaged. Declarations and metadata are supported for the first two of these. The others are the subject of further work; revised versions of this package can be expected in later years.

2 Usage

The package can be loaded with the command:

\usepackage[⟨*options*⟩]{pdfx}

where the options are as follows.

2.1 Options

2.1.1 PDF/A options

PDF/A is an ISO standard [3–5] intended for long-term archiving of electronic documents. It therefore emphasizes self-containedness and reproducibility, as well as machine-readable metadata. The PDF/A standard has three conformance levels "a", "b", and "u". Level "a" is the strictest, and is not yet fully implemented by the pdfx package. Conformance level "u" has the same requirements as level "b", but

with the additional requirement that all text in the document must have a Unicode mapping. The `pdfx` package produces such Unicode mappings even in level "b" files. The standard also has three different versions 1, 2, and 3, which were standardized in 2005, 2011, and 2012, respectively. Earlier versions contain a subset of the features of later versions, so for maximum portability, it is preferable to use a lower-numbered version. There is no conformance level "u" in version 1 of the standard. For many typical uses of PDF/A, it is sufficient to use PDF/A-1b.

- `a-1a`: generate PDF/A-1a. Experimental, not fully implemented.
- `a-1b`: generate PDF/A-1b.
- `a-2a`: generate PDF/A-2a. Experimental, not fully implemented.
- `a-2b`: generate PDF/A-2b.
- `a-2u`: generate PDF/A-2u.
- `a-3a`: generate PDF/A-3a. Experimental, not fully implemented.
- `a-3b`: generate PDF/A-3b.
- `a-3u`: generate PDF/A-3u.

By 'Experimental, not fully implemented' we mean primarily that the document structure, as required for 'Tagged PDF', is not handled by this package. Using other pdfTEX-based software that *is* capable of producing such complete tagging, conforming documents can indeed be produced.

2.1.2 PDF/E options

PDF/E is an ISO standard intended for documents used in engineering workflows. There is only one version of the PDF/E standard so far, and it is called PDF/E-1.

- `e-1`: generate PDF/E-1.

2.1.3 PDF/VT options

PDF/VT is an ISO standard intended as an exchange format for variable and transactional printing, and is an extension of the PDF/X-4 standard. The standard specifies three PDF/VT conformance levels. Level 1 is for single-file exchange, level 2 is for multi-file exchange, and level 2s is for streamed delivery. Currently, none of the PDF/VT conformance levels are fully implemented by the `pdfx` package.

- `vt-1`: generate PDF/VT-1. Experimental, not fully implemented.
- `vt-2`: generate PDF/VT-2. Experimental, not fully implemented.
- `vt-2s`: generate PDF/VT-2s. Experimental, not fully implemented.

By 'Experimental, not fully implemented' here we mean primarily that the structuring of a document into '/DPart' sections, as Form XObjects, is not handled by this package. This *is* possible with current pdfTEX software, but not yet in a way that lends itself easily to full automation, due to requirements of knowing the internal object number of certain internal PDF constructs. All other aspects — PDFInfo declaration, metadata and color profile — of the PDF/VT variants are correctly handled.

2.1.4 PDF/X options

PDF/X is an ISO standard intended for graphics interchange. It emphasizes printing-related requirements, such as embedded fonts and color profiles. The PDF/X standard has a large number of variants and conformance levels. The basic variants are known as X-1, X-1a, X-3, X-4, and X-5. (A revised version of the X-2 standard was published in 2003, but withdrawn as an ISO standard in 2011, basically due to lack of interest in using it.) The PDF/X-1a standard exists in revisions of 2001 and 2003, the PDF/X-3 standard exists in revisions of 2002 and 2003, and the PDF/X-4 and PDF/X-5 standards exist in revisions of 2008 and 2010. Moreover, some of these standards have a 'p' version, which permits the use of an externally supplied color profile (instead of an embedded one), and/or a 'g' version, which permits the use of external graphical content. Moreover, PDF/X-5 has an 'n' version, which extends PDF/X-4p by permitting additional color spaces other than grayscale, RGB, and CMYK. For many typical uses of PDF/X, it is sufficient to use PDF/X-1a.

- `x-1`: generate PDF/X-1.
- `x-1a`: generate PDF/X-1a. Options `x-1a1` and `x-1a3` are also available to specify PDF/X-1a:2001 or PDF/X-1a:2003 explicitly.
- `x-3`: generate PDF/X-3. Options `x-302` and `x-303` are also available to specify PDF/X-3:2002 or PDF/X-3:2003 explicitly.
- `x-4`: generate PDF/X-4. Options `x-408` and `x-410` are also available to specify PDF/X-4:2008 or PDF/X-4:2010 explicitly.
- `x-4p`: generate PDF/X-4p. Options `x-4p08` and `x-4p10` are also available to specify PDF/X-4p:2008 or PDF/X-4p:2010 explicitly.
- `x-5g`: generate PDF/X-5g. Options `x-5g08` and `x-5g10` are also available to specify PDF/X-5g:2008 or PDF/X-5g:2010 explicitly.
- `x-5n`: generate PDF/X-5n. Options `x-5n08` and `x-5n10` are also available to specify PDF/X-5n:2008 or PDF/X-5n:2010 explicitly. Experimental, not fully implemented.

- `x-5pg`: generate PDF/X-5pg. Options `x-5pg08` and `x-5pg10` are also available to specify PDF/X-5pg:2008 or PDF/X-5pg:2010 explicitly.

2.1.5 Other options

These options are experimental and should not normally be used.

- `useBOM`: generate an explicit UTF-8 byte-order marker in the embedded XMP metadata, and make the XMP packet writable. Neither of these features are required by the PDF/A standard, but there exist some PDF/A validators (reportedly `validatepdfa.com`) that seem to require them. Note: the implementation of this feature is experimental and may break with future updates to the `xmpincl` package.
- `noBOM`: do not generate the optional byte-order marker (default).
- `pdf13`: use PDF 1.3, overriding the version specified by the applicable standard. This may produce a non-standard-conforming PDF file.
- `pdf14`: use PDF 1.4, overriding the version specified by the applicable standard. This may produce a non-standard-conforming PDF file.
- `pdf15`: use PDF 1.5, overriding the version specified by the applicable standard. This may produce a non-standard-conforming PDF file.
- `pdf16`: use PDF 1.6, overriding the version specified by the applicable standard. This may produce a non-standard-conforming PDF file.
- `pdf17`: use PDF 1.7, overriding the version specified by the applicable standard. This may produce a non-standard-conforming PDF file.

2.2 Data file for metadata

As mentioned above, standards-compliant PDF documents require metadata to be included. The `pdfx` package expects the metadata to be supplied in a special data file called `\jobname.xmpdata`. Here, `\jobname` is usually the basename of the document's main `.tex` file. For example, if your document source is `main.tex`, then the metadata must be in a file called `main.xmpdata`. None of the individual metadata fields are mandatory, but for most documents, it makes sense to define at least the title and the author. Here is an example of a short `.xmpdata` file:

```
\Title{Baking through the ages}
\Author{A. Baker\sep C. Kneader}
\Keywords{cookies\sep muffins\sep cakes}
\Publisher{Baking International}
```

Please note that multiple authors and keywords have been separated by `\sep`. The `\sep` macro is only permitted in the `\Author`, `\Keywords`, and `\Publisher` fields.

After processing, the local directory contains a file named such as `pdfa.xmpi` or `pdfx.xmpi` according to the PDF variant required. This file is the complete XMP metadata packet. It can be checked for validity using an online validator, such as: `http://www.pdflib.com/knowledge-base/xmp-metadata/free-xmp-validator`

2.3 List of supported metadata fields

Here is a complete list of user-definable metadata fields currently supported, and their meanings. More may be added in the future. These commands can only be used in the `.xmpdata` file.

2.3.1 General information:

- `\Author`: the document's human author. Separate multiple authors with `\sep`.
- `\Title`: the document's title.
- `\Keywords`: list of keywords, separated with `\sep`.
- `\Subject`: the abstract.
- `\Publisher`: the publisher.

2.3.2 Copyright information:

- `\Copyright`: a copyright statement.
- `\CopyrightURL`: location of a web page describing the owner and/or rights statement for this document.
- `\Copyrighted`: "True" if the document is copyrighted, and "False" if it isn't. This is automatically set to "True" if either `\Copyright` or `\CopyrightURL` is specified, but can be overridden. For example, if the copyright statement is "Public Domain", this should be set to "False".

2.3.3 Publication information:

- `\PublicationType`: The type of publication. If defined, must be one of book, catalog, feed, journal, magazine, manual, newsletter, pamphlet. This is automatically set to "journal" if `\Journaltitle` is specified, but can be overridden.
- `\Journaltitle`: The title of the journal where the document was published.
- `\Journalnumber`: The ISSN for the publication in which the document was published.
- `\Volume`: Journal volume.
- `\Issue`: Journal issue/number.
- `\Firstpage`: First page number of the published version of the document.

- \Lastpage: Last page number of the published version of the document.
- \Doi: Digital Object Identifier (DOI) for the document, without the leading "doi:".
- \CoverDisplayDate: Date on the cover of the journal issue, as a human-readable text string.
- \CoverDate: Date on the cover of the journal issue, in a format suitable for storing in a database field with a "date" data type.

2.4 Symbols permitted in metadata

Within the metadata, all printable ASCII characters except \, {, }, and % represent themselves. Also, all printable Unicode characters from the basic multilingual plane (i.e., up to code point U+FFFF) can be used directly with the UTF-8 encoding. (Encodings other than UTF-8 are not currently supported in the metadata). Consecutive whitespace characters are combined into a single space. Whitespace after a macro such as \copyright, \backslash, or \sep is ignored. Blank lines are not permitted. Moreover, the following markup can be used:

- "\ ": a literal space (for example after a macro)
- \%: a literal %
- \{: a literal {
- \}: a literal }
- \backslash: a literal backslash, \
- \copyright: the copyright symbol, ©

The macro \sep is only permitted within \Author, \Keywords, and \Publisher. Its intention is to separate multiple authors, keywords, etc. However, for validation purposes, multiple authors and keywords must not be truly separated. The package takes care of this, even when \sep is used.

It turns out that other TEX macros can be used, provided the author is very careful and does not ask for too-complicated TEX or LATEX expansions into internal commands or non-character primitives; basically just accents, macros for Latin-based special characters, and simple textual replacements, perhaps with a simple parameter. A special macro

```
\pdfxEnableCommands{...}
```

is provided to help resolve difficulties that may arise.

For example, \pdfxEnableCommands is needed with the name of one of our authors, Hàn Thế Thành, due to the doubly-accented letter ế. It is usual to define a macro such as:

```
\def\thanh{H\‘an Th\’{\^e} Thanh}
```

In previous versions of the pdfx package, use of such a macro within the .xmpdata file, in the Copyright information say, could result in the accent macros expanding into internal primitives, such as

```
H\unhbox \voidb@x \bgroup
 \let \unhbox \voidb@x \setbox ...
```

going on for many lines. This clearly has no place within the XMP metadata. To get around this, one could try using simplified macro definitions

```
\pdfxEnableCommands{
  \def\‘#1{#1^^cc^80}
  \def\’#1{#1^^cc^81}
  \def\^#1{#1^^cc^82}}
```

where the ^^cc^80, ^^cc^81, ^^cc^82 cause TEX to generate the correct UTF-8 bytes for 'combining accent' characters.

This works fine for metadata fields that appear just in the XMP packet. However, it is not sufficient for the PDF /Author key, which must exactly match with the dc:creator metadata element. What is needed instead is

```
\pdfxEnableCommands{
  \def\thanh{H^^c3^^a0n Th\eee Thanh}
  \def\eee{^^c3^^aa^^cc^^81 }}
```

or the above with 'à' typed directly as UTF-8 instead of ^^c3^^a0 and 'ê' in UTF-8 for ^^c3^^aa. The reason for this is due to the \pdfstringdef command, which constructs the accented Latin letters as single combined characters à and ê, without resorting to combining accents, wherever possible. If the metadata does not have the same, irrespective of Unicode normalisation, then validation fails.

With the latest version of the pdfx package, such difficulties have been overcome, at least for characters used in Western European, Latin-based languages. The input encoding used when reading the .xmpdata file now includes interpretations of TEX's usual accent commands to produce the required UTF-8 byte sequences. Work is ongoing to extend this input encoding to additional languages (e.g., extended Latin, Cyrillic, Greek, etc.). A significant portion of the Unicode Basic Plane characters can be covered this way. Modules could even be provided for CJK character sets and mathematical symbols, etc. However, this can become memory intensive, so significant testing will be required before this becomes a standard part of the pdfx package.

2.5 Color profiles

Most standards-compliant PDF documents require a *color profile* to be embedded within the file. In a nutshell, such a profile determines precisely how the colors used in the document will be rendered when printed to a physical medium. This can be used to ensure that the document will look exactly the same, even when it is printed on different printers, with different paper types, etc. The inclusion of a color

profile is necessary to make the document completely self-contained.

Since most LaTeX users are not graphics professionals and are not particularly picky about colors, the pdfx package includes default profiles that will be included when nothing else is specified. Therefore, the average user doesn't have to do anything special about color.

Users who wish to use a specific color profile can do so by including a \setRGBcolorprofile or \setCMYKcolorprofile command in the .xmpdata file. Note that PDF/A and PDF/E accept an RGB color profile, while PDF/X and PDF/VT require a CMYK color profile. Use the following commands to specify an RGB or CMYK color profile, respectively:

\setRGBcolorprofile{⟨*filename*⟩}
{⟨*identifier*⟩}{⟨*info string*⟩}{⟨*registry URL*⟩}

\setCMYKcolorprofile{⟨*filename*⟩}
{⟨*output intent*⟩}{⟨*identifier*⟩}{⟨*registry URL*⟩}

Within the arguments of these macros, the characters <, >, &, ^, _, #, $, and ~ can be used as themselves, but % must be escaped as \%. The defaults are:

```
\setRGBcolorprofile
   {sRGB_IEC61966-2-1_black_scaled.icc}
   {sRGB_IEC61966-2-1_black_scaled}
   {sRGB IEC61966 v2.1 with black scaling}
   {http://www.color.org}

\setCMYKcolorprofile
   {coated_FOGRA39L_argl.icc}
   {Coated FOGRA39}
   {FOGRA39 (ISO Coated v2 300\% (ECI))}
   {http://www.argyllcms.com/}
```

Some color profile files may be obtained from the International Color Consortium; please see http://www.color.org/iccprofile.xalter.

Alternatively, color profiles are shipped with many Adobe software applications; these are then available for use also with non-Adobe software. Now the pdfx package includes coding to streamline inclusion of these profiles in PDF documents, or to specify them as 'external' profiles, with PDF/X-4p and PDF/X-5pg variants. Two files AdobeColorProfiles.tex and AdobeExternalProfiles.tex are distributed with the pdfx package. The latter is for use with PDF/X-4p and PDF/X-5pg, which do not require color profiles to be embedded, while the former can be used with other PDF/X variants. Both define commands to use color profiles as shown in Table 1.

As of the time of writing, only the first six of these result in PDFs which can validate with external profiles (i.e., for PDF/X-4p and PDF/X-5pg) using current versions of Adobe Acrobat Pro software. It is unclear whether the others (incl. \IFRAXXVI and \IFRAXXX) fail due to incorrect data or problems in the validation software. All but those last two can be used for valid embedded profiles, providing the corresponding files can be found. The following macro is used to set the (absolute or relative) path, on the local operating system, to the location of color profile files.

Table 1: Commands for various color profiles.

Command	Name of Profile
\FOGRAXXXIX	Coated FOGRA39 (ISO 12647-2:2004)
\SWOPCGATSI	U.S. Web Coated (SWOP) v2
\JapanColorMMICoated	Japan Color 2001 Coated
\JapanColorMMIUncoated	Japan Color 2001 Uncoated
\JapanColorMMIINewspaper	Japan Color 2002 Newspaper
\JapanWebCoatedAd	Japan Web Coated (Ad)
\CoatedGRACoL	Coated GRACoL 2006 (ISO 12647-2:2004)
\SNAPCGATSII	CGATS TR 002
\SWOPCGATSIII	CGATS TR 003
\SWOPCGATSV	CGATS TR 005
\ISOWebCoated	Web Coated FOGRA28 (ISO 12647-2:2004)
\ISOCoatedECI	ISO Coated v2 (ECI)
\CoatedFOGRA	Coated FOGRA27 (ISO 12647-2:2004)
\WebCoatedFOGRA	Web Coated FOGRA28 (ISO 12647-2:2004)
\UncoatedFOGRA	Uncoated FOGRA29 (ISO 12647-2:2004)
\IFRAXXVI	ISOnewspaper26v4 ISO/DIS 12647-3:2004
\IFRAXXX	ISOnewspaper30v4 ISO/DIS 12647-3:2004

\pdfxSetColorProfileDir
{⟨*path to Adobe color profiles*⟩}

2.6 Notes on internal representation of metadata

Within the PDF file, metadata is deposited in two places: some data goes into the native PDF /Info dictionary, and some data goes into an XMP packet stored separately within the file. XMP is Adobe's Extensible Metadata Platform, and is an XML-based format. See the Adobe XMP Development Center (http://www.adobe.com/devnet/xmp) for more exhaustive information about XMP. An XMP Toolkit SDK which supports the GNU/Linux, Macintosh and Windows operating systems is also provided under the modified BSD license.

Some of the metadata, such as the author, title, and keywords, are stored *both* in the XMP packet and in the /Info dictionary. For the resulting file to be standards-compliant, the two copies of the data must

be identical. All of this is taken care of automatically by the pdfx package.

In principle, users can resort to alternate ways to create an XMP file for inclusion in PDF. In this case, users should create a file pdfa.xmp or pdfx.xmp (etc., depending on the PDF flavor) containing the pre-defined data. However, this is an error-prone process and is not recommended for most users. If there is a particular field of metadata that you need and that is not currently supported, please contact the authors.

pdfx makes use of the xmpincl package to include xmp data into the PDF. The documentation of the xmpincl package may help interested users to understand the process of xmp data inclusion.

2.7 Tutorials and technical notes

A tutorial with step-by-step instructions for generating PDF/A files is at:
http://www.mathstat.dal.ca/~selinger/pdfa

Some technical notes about production problems that authors have encountered in generating PDF/A-compliant documents are at:
http://support.river-valley.com/wiki/index.php?title=Generating_PDF/A_compliant_PDFs_from_pdftex

3 Installing

The pdfx package is available on CTAN as usual, via http://ctan.org/pkg/pdfx. It is also included in TEX distributions such as TEX Live and MiKTEX.

3.1 Limitations and dependencies

pdfx.sty works with pdfTEX and also LuaTEX. It further depends on the following other packages:

1. xmpincl for insertion of metadata into PDF.
2. hyperref for hyperlinking, bookmarks, etc.
3. glyphtounicode.tex for mapping glyph names to corresponding Unicode.

3.2 Files included

The following files are included in the package. Some are created from pdfx.dtx, following the Makefile.

3.2.1 Package files

- pdfx.sty: main package file (from pdfx.dtx).
- pdfa.xmp: specimen xmp template for PDF/A.
- pdfe.xmp: specimen xmp template for PDF/E.
- pdfvt.xmp: specimen xmp template for PDF/VT.
- pdfx.xmp: specimen xmp template for PDF/X.
- 8bit.def: custom input encoding.
- l8uenc.def: input encoding macro declarations.
- glyphtounicode-cmr.tex: maps glyph names to corresponding Unicode for Computer Modern and other TEX-specific fonts.
- coated_FOGRA39L_argl.icc: CMYK color profile (freely distributable).
- sRGB_IEC61966-2-1_black_scaled.icc: RGB color profile (freely distributable).
- ICC_LICENSE.txt: license for the color profiles.
- AdobeColorProfiles.tex: macros for inclusion of Adobe-supplied color profiles.
- AdobeExternalProfiles.tex: macros for use of external color profiles.

3.2.2 Documentation

- README: usual top-level information.
- manifest.txt: file list.
- sample.tex, sample.xmpdata: a sample file with sample metadata.
- small2e-pdfx.tex, small2e-pdfx.xmpdata: another sample file with sample metadata.

3.2.3 Sources

- src/pdfx.dtx: literate source combining code and documentation.
- src/pdfx.ins: installer batch file.
- src/rvdtx.sty: used by pdfx.dtx.
- src/Makefile: a Makefile for building the documentation.

3.3 Licensing and contact

The package is released under the LATEX Project Public License. Bug reports, suggestions, feature requests, etc., may be sent to any of the authors at the addresses below.

Bibliography

[1] Adobe Systems Inc.; PDF Reference 1.7, November 2006. Also available as [6]. http://www.adobe.com/devnet/pdf/pdf_reference.html.

[2] Dublin Core Metadata Element Set, Version 1.1, October 2010. http://dublincore.org/documents/dces/

[3] ISO 19005-1:2005; Document Management — Electronic document file format for long term preservation — Part 1: Use of PDF 1.4 (PDF/A-1); Technical Committee ISO/TC 171/SC 2 (Sept. 2005). Revisions via Corrigenda: ISO 19005-1:2005/Cor 1:2007 (March 2007); ISO 19005-1:2005/Cor 2:2011 (Dec. 2011). http://www.iso.org/iso/catalogue_detail?csnumber=38920.

[4] ISO 19005-2:2011; Document Management — Electronic document file format for long term preservation — Part 2: Use of ISO 32000-1 (PDF/A-2); Technical Committee ISO/TC 171/SC 2 (June 2011). http://www.iso.org/iso/catalogue_detail?csnumber=50655.

[5] ISO 19005-3:2012; Document Management — Electronic document file format for long term preservation — Part 3: Use of ISO 32000-1 with support for embedded files (PDF/A-3); Technical Committee ISO/TC 171/SC 2 (October 2012). http://www.iso.org/iso/catalogue_detail?csnumber=57229.

[6] ISO 32000-1:2008; Document management — Portable document format (PDF 1.7); Technical Committee ISO/TC 171/SC 2 (July 2008). Also available as [1]. http://www.iso.org/iso/catalogue_detail?csnumber=51502.

[7] ISO 32000-2-20140220; Document management — Portable document format — Part 2: PDF 2.0; Technical Committee ISO/TC 171/SC 2, in draft form (Feb. 2014).

[8] ISO 24517-1:2008; Document Management — Engineering document format using PDF — Part 1: Use of PDF 1.6 (PDF/E-1); Technical Committee ISO/TC 171/SC 2 (May 2008). http://www.iso.org/iso/catalogue_detail?csnumber=42274.

[9] ISO 14289-1:2012; Document management applications — Electronic document file format enhancement for accessibility — Part 1: Use of ISO 32000-1 (PDFUA-1); Technical Committee ISO/TC 171/SC 2 (July 2012). http://www.iso.org/iso/catalogue_detail.htm?csnumber=54564.
Revised as ISO 14289-1:2014 (December 2014): http://www.iso.org/iso/catalogue_detail.htm?csnumber=64599.

[10] PDFUA Technical Implementation Guide: Understanding ISO 14289-1 (PDFUA-1). AIIM Global Community of Information Professionals. http://www.aiim.org/Research-and-Publications/standards/committees/PDFUA/Technical-Implementation-Guide.

[11] ISO 16612-2:2010; Graphic technology — Variable data exchange — Part 2: Using PDF/X-4 and PDF/X-5 (PDFVT-1 and PDFVT-2). Technical Committee ISO/TC 130 (December 2005). http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=38013.

[12] ISO 15930-1:2001; Graphic technology — Prepress digital data exchange — Use of PDF — Part 1: Complete exchange using CMYK data (PDF/X-1 and PDF/X-1a). Technical Committee ISO/TC 130 (December 2001). http://www.iso.org/iso/catalogue_detail.htm?csnumber=29061.

[13] ISO 15930-3:2002; Graphic technology — Prepress digital data exchange — Use of PDF — Part 3: Complete exchange suitable for colour-managed workflows (PDF/X-3). Technical Committee ISO/TC 130 (September 2002). http://www.iso.org/iso/catalogue_detail.htm?csnumber=34941.

[14] ISO 15930-4:2003; Graphic technology — Prepress digital data exchange — Use of PDF — Part 4: Complete exchange of CMYK and spot colour printing data using PDF 1.4 (PDF/X-1a). Technical Committee ISO/TC 130 (December 2003). http://www.iso.org/iso/catalogue_detail.htm?csnumber=39938.

[15] ISO 15930-6:2003; Graphic technology — Prepress digital data exchange — Use of PDF — Part 6: Complete exchange of printing data suitable for colour-managed workflows using PDF 1.4 (PDF/X-3). Technical Committee ISO/TC 130 (December 2003). http://www.iso.org/iso/catalogue_detail.htm?csnumber=39940.

[16] ISO 15930-7:2010; Graphic technology — Prepress digital data exchange — Use of PDF — Part 7: Complete exchange of printing data (PDF/X-4) and partial exchange of printing data with external profile reference (PDF/X-4p) using PDF 1.6. Technical Committee ISO/TC 130 (July 2010). http://www.iso.org/iso/catalogue_detail.htm?csnumber=55843.

[17] ISO 15930-8:2010; Graphic technology — Prepress digital data exchange — Use of PDF — Part 8: Partial exchange of printing data using PDF 1.6 (PDF/X-5). Technical Committee ISO/TC 130 (July 2010). http://www.iso.org/iso/catalogue_detail.htm?csnumber=55844.
Revision via Corrigendum: ISO 15930-8:2010/Cor 1:2011 (August 2011). http://www.iso.org/iso/catalogue_detail.htm?csnumber=60210.

[18] ISO 16684-1:2012; Graphic technology — Extensible metadata platform (XMP) specification — Part 1: Data model, serialization and core properties. Technical Committee ISO/TC 130 (February 2012). http://www.iso.org/iso/catalogue_detail.htm?csnumber=57421.

◇ C. V. Radhakrishnan
River Valley Technologies
Trivandrum, India 695014
cvr (at) river-valley dot org
http://cvr.cc/

◇ Hàn Thế Thành
River Valley Technologies
Trivandrum, India 695014
thanh (at) river-valley dot org

◇ Ross Moore
Mathematics Department
Macquarie University
Australia 2109
ross (at) mq dot edu dot au
http://maths.mq.edu.au/~ross/

◇ Peter Selinger
Department of Mathematics and Statistics
Dalhousie University
Halifax, Nova Scotia B3H 4R2
Canada
selinger (at) mathstat dot dal dot ca
http://www.mathstat.dal.ca/~selinger/

TEXShop's key bindings vs. macros vs. command completion

Herbert Schulz

Abstract

A workshop was held at TUG'15 about some features of TEXShop that are sometimes confused. The following article is based on the topics discussed, but we'll start with some general information about TEXShop.

1 Introduction

TEXShop is a "front end" for TEX on Mac OS X. As such it allows the user to create and edit TEX source files, interact with an installed TEX distribution (e.g., typeset the source file) and finally preview the final PDF file. It also allows the user to go back and forth between preview and source.

Over the years TEXShop has added many features. Some of them are obvious and are meant to help a novice get started. Others are a bit more subtle in their use and the underlying power of these features needs to be coaxed out.

2 Three features

There are three features of TEXShop which often get mixed up: Key Bindings (at one time called Auto-Completion), Macros, and Command Completion. Although they share similar features it is possible to discern a difference between them and decide when each is most useful.

Key Bindings assign a sequence of keystrokes to the press of a single key; e.g., typing '_' produces '_{...|}' (where ...| is any selected text followed by the insertion point — the place where newly-typed text is inserted) or typing '≤' (Opt-, with the English keyboard layout) produces '\leq'.

Macros can also insert simple text and be given a keyboard shortcut (that always uses Cmd plus other keys) but are most useful when attached to AppleScript programs so they can do special processing of source text, etc.

Command Completion (*NOT* to be confused with Auto-Completion) allows you to type a partial command or short abbreviation and, when a trigger key is pressed (Esc by default but it can be set to Tab), have the text so far expanded into a full command or even a full environment structure.

Details follow.

3 Key Bindings

Key Bindings are enabled in general by checking TeXShop → Preferences → Source → Key Bindings. Of course they can be turned off by un-checking that preference setting. They can be turned on/off for a given editing session by using the Source → Key Bindings → Toggle On/Off menu item (a check mark means it's on).

You can see view, edit, add to, and remove from the list of Key Bindings by starting up the Key Bindings Editor via the menu item Source → Key Bindings → Edit Key Bindings File....

3.1 Notes on Key Bindings

- Inserting a \ before typing the key negates the expansion; e.g., pressing \ and then _ produces _, as it should.
- Key Bindings cannot be created for characters produced by multi-key sequences; e.g., Opt-e *followed by* e produces é on the English keyboard layout which *cannot* be set to produce \'e as a Key Binding. The initial Opt-e is usually called a dead key since it doesn't produce an on-screen character by itself and *must* be followed by another character.
- When creating a Key Binding using #SEL# or #INS# in the replacement text will place any selected text or the insertion point at that location respectively.
- Let's emphasize that Key Bindings *always* expand to the assigned text (unless escaped with \). There may be times when you don't wish an expansion of that keystroke. You should create a Macro to do that for you.

4 Macros

The Macros menu contains a fairly large number of pre-defined macros. You can create another macro with the Macro Editor, via Macros → Open Macro Editor. Macros can be added either by copying text into a newly created macro or by adding a macro file (with extension plist) using Macros → Add macros from file (only visible and available when the Macro Editor is active).

The order of appearance of the macros in the Macros menu can also be changed by simply moving them around on the left panel of the Macro Editor.

4.1 Notes on Macros

- Text to which you wish to assign a Cmd-based keyboard shortcut is best created using a Macro rather than a Key Binding; e.g., there is already a Macro that takes selected text and sets it in boldface (using \textbf) and you can assign the Cmd-B keyboard shortcut to that Macro in the Macro Editor.

- When creating a text macro using `#SEL#` or `#INS#` in the replacement text will place any selected text or the insertion point at that location respectively.

5 Command Completion

For Command Completion you enter a partial command name or a short abbreviation, press a trigger key (Esc or Tab, mentioned above, if set in TeXShop → Preferences → Source → Command Completion Triggered By:) and it gets expanded. E.g., enter

```
\sec
```

on a new line and press the trigger (Esc or ...) and you get

```
\section{•}
```

(where • is a selected bullet — called a Mark in Command Completion parlance — so simply typing will replace that Mark with your text. There can be more than one match for a given input; if you press the trigger again (without entering text) you get

```
\section*{•}
```

and another press of the trigger gives

```
\section[•]{•}
```

for separate section titles in the toc and the document. In the last case there is a second Mark (•) for the second argument. After entering the toc section title you jump to and select the next Mark by using Source → Command Completion → Marks → Next Mark (Ctl-Cmd-F) so you can immediately start typing the section title for that document.

Better yet are abbreviations. E.g., type

```
\benu
```

(abbreviations for environments always start with a 'b') and press the trigger key to get

```
\begin{enumerate}
\item
•
\end{enumerate}•
```

... ready to enter the first item.

Then to get a new `\item` simply type

```
\it
```

on a new line and the trigger to get

```
\item
•
```

To get to the very end of the enumerate environment use Ctl-Cmd-F to select the Mark at the end of the environment where simply typing Return will remove that mark and move to the next line.

5.1 Notes on Command Completion

- Command Completion *replaces* any selected text by the expansion. This is unlike Key Bindings and Macros, which can be written to include the selected text in the final result of their actions.
- The easiest way to learn how to create your own command completions or abbreviations is to examine the Command Completion File using Source → Command Completion → Edit Command Completion File....
- When creating a completion or abbreviation for Command Completion you cannot use `#SEL#` since any selection will be replaced by the completion. Using `#INS#` in the replacement text will place the insertion point at that location respectively. You can also use two copies of `#INS#` and any text between them will be selected; e.g., this is useful for first arguments which should be a selected Mark, •, created with `#INS#•#INS#`. Use Cmd-8 to produce the Mark because Key Bindings, if active, will replace a typed • with `\textbullet`.

6 In closing

The current version of this document (labeled as TeXShopFeatureConfusion), and many other TEX- and TEXShop-related items, are available at the DropBox url below. The TeXShopTips document there may be of particular interest.

◇ Herbert Schulz
herbs2 (at) mac dot com
https://dl.dropboxusercontent.com/u/10932738/index.html

TEX as a three-stage rocket: Cookie-cutter page breaking

S.K. Venkatesan

Abstract

We trace the progression of technological developments from the early days of computers and how TEX has been able to maneuver through all these changes. A major challenge now arises from the ubiquitous browser and HTML5. We propose that TEX reinvent itself as a three-stage engine: 1) paragraph generation; 2) creation of very long scroll page; 3) page-breaking through a simple cookie-cutter process. With these relatively small adjustments TEX and its typographic nuances could be liberated into the exciting wide open spaces of the World Wide Web.

1 Introduction

Donald Knuth arrived in an age in the United States when universities were making important changes to the digital landscape, notably including Stanford University. The arrival of big capital in software development had already created grave doubts in the mind of the creator of Emacs, Richard Stallman. It is indeed a great achievement that an interactive system with its own special markup, especially for mathematics, introduced in 1979, has been used for 36 years, despite all the advancement of big capital and its great innovations. Knuth's use of the dollar sign to create mathematics of priceless quality is an amusing comment on poor quality commercial typography. Knuth's recent announcement of iTEX [1] was also an apt mockery of the pompous ways of big capital.

However, there have been great changes in the computing landscape. First there were separate non-interactive punch card devices which gave way to interactive command-line interfaces. Second there were automated printing devices and then non-interactive graphics plotting devices. Finally out of all these evolved the modern hybrids: desktops, laptops, tablets, e-readers, mobile phones, etc. It is indeed a great achievement that TEX has survived all these incredible changes in devices and systems. TEX is still probably the best text format to SMS maths in mobiles!

2 SGML, HTML and the browser

Big capital then had its own innovation, the SGML DTD. It could have used a backslash syntax similar to TEX or a parenthetical syntax similar to Lisp; in fact the SGML language was so general that it could accommodate any syntax. Knuth, the mathematician, would not have agreed to use the less-than symbol for markup but that is how it is in the world of business and there is similarly no way a business man would agree to use the dollar sign for markup. Knuth's nice choice of the rarely used backslash for starting macros was replaced with the commonly-used ampersand symbol, which now has to be displayed by escaping it as an entity. However, SGML had an additional feature that TEX didn't have: an agreed-on grammar (EBNF) for parsing, which made it popular with its evangelists. By the late 1980s, SGML+DTD became the industry standard for the large publishers as it could now be validated! However, SGML and the DSSSL stylesheet language for SGML was unwieldy, and it was not easy to process SGML for typesetting. Many commercial companies created SGML-friendly systems with some daring to even implement the unwieldy DSSSL.

The arrival of the Internet produced HTML through Tim Berners-Lee at CERN, a simple implementation of many futuristic concepts of Ted Nelson's Xanadu [2]. With the introduction of the Mosaic browser, the hyperlinking between documents created a new revolution, creating the web as we see it now. It was no more the stale references at the end of a document which one has to look-up in the library; now you can click at links and travel to the linked documents! This led to greater democratisation of knowledge, especially with the arrival of Wikipedia and other open access publishing models, that has continued to expand the open World Wide Web.

Print was still used a lot in those days and it never dawned on anyone at that time that the advent of the browser would slowly bring the end to printing on paper. Internet browsers had a scroll bar that could be scrolled down to read the whole document. Ancient cloth scrolls now had their new avatar. The creation of codex pages that had become the vogue from the 14th century in Europe with the Gutenberg press in Germany is now in the process of being undone.

Of course, the concept of pages is still popular in HTML eBook renderers also. Adobe PostScript and PDF are still popular page models that are used for viewing pages, as the codex refuses to die in the annals of history. The invention of electronic e-paper or e-ink show how friendly glare-free paper is! As the page numbers disappear in eBook Readers like Kindle, they are replaced by the percentage of the document travelled by the reader. Of course, a page number could be generated automatically based on the device width and height, but that would not be a device-independent value any more as in PDF.

3 TEX in a very long browser page

IBM Tech Explorer Hypermedia Browser was one of the early attempts at creating a browser for LATEX markup. It was implemented as plugins for both Netscape and Internet Explorer browsers. Although it only implemented limited features of TEX, it was nevertheless quite useful for viewing scientific documents. Back in 1998, Springer-Verlag's *Linear Functions and Matrix Theory* by author Bill Jacob was one of the first publications that had the privilege of being distributed this way.

Illusions of Java being the platform of the future have faded in recent years, especially after abandonment of the $\mathcal{NTS}$ (New Typesetting System). Mainstream development has now shifted to JavaScript with many new implementations of TEX in JavaScript. The JsMath package has been extended by the MathJax team that has now been actively implementing math bits of TEX in the browser. KaTEX is another excellent implementation of TEX math in JavaScript by the Khan academy, which produces popular lectures on all academic topics. The LaTeXMath.js package tried to implement a complete LATEX document. There is also a complete JavaScript port of pdfTEX (texlive.js) that is now production ready, but it produces PDF on client-side, not web pages!

Boris Veytsman [3] tried to create very long PDF pages using TEX. He used output routines to get the desired long scroll page. However, the maximum limit that TEX can produce is 5 meters! This is also the limit in the PDF specification, a limit that is not easy to relax in PDF, unless we use special units. Thus it is clear that we have to think beyond PDF and into the world of browsers.

The arbitrarily long fluid web pages have the advantage that they can flow into different sized devices, from small hand-held mobiles to wide Desktop screens. The line-breaking algorithms of the browsers have been investigated by the author and compared with the quality of line-breaking that is generally expected out of the TEX system. In the next section we will consider this aspect in detail.

4 Line-breaking: TEX versus browsers

Modern typesetting applications and TEX differ in many details of the line-breaking algorithms. The author has been exposed to a wide variety of commercial WYSIWYG typesetting applications, which make some compromise to render pages faster. However, it is quite possible to produce quality line-breaking from these applications by controlling the default settings of these typesetting systems.

In Figs. 1 and 2 we show some sample paragraphs produced by XETEX and the Firefox browser to show the differences.

As we can see from this output that by making adjustments in CSS, it is possible to obtain output quite close in quality to that produced by TEX or even surpass it in some cases.

5 Footnotes

Users of eBook reader devices have always been uncomfortable with footnotes becoming endnotes, so that they have to scroll back and forth from its citation, making it inconvenient. The IBM Tech Explorer rendered it as a pop-up notes in those days but there have been other solutions such as sidenotes or a pop-balloon.

The author's own solution for this case[1] has been a bottom rule at the end of the paragraph where the citation of footnote appears.

[1] Footnote below a paragraph

6 Floats: figures and tables

In a bottomless scroll page, where do we place floats, i.e., figures and tables? This again has various "magical" solutions, like pop-ups. The author's simple solution in these cases is again to place it after the paragraph in which it is cited. Of course, it is ideal if the writer of the document indicates the exact location for these floats. Figures and tables can remain at whatever size they are intended, but we can restrict the maximum size to the size of the device, and/or we can add a scroll bar if it is too large, as in the case of wide tables.

7 Pagination as part of a third stage in a TEX browser

It is appropriate at this juncture to create a TEX DVI browser that produces pages as long as there is content without page breaks. The first stage in this process is the module that produces H&J of each paragraph of text. Paragraph creation should be a separate module, as it is much more efficient for an AJAX application to update changes in a paragraph without having to repaint the entire document. Next, we apply the vertical glues, footnotes, floats as mentioned in earlier sections to get the long scroll page.

In a simple streaming-down process it is now possible to automatically decide the page breaks and move the floats one-by-one as we paginate. The floats have already been placed close to their citation in the scroll page, so in a single sweep it is possible to place them as we paginate. Widows and orphans can be controlled by appropriately controlling potentially valid line breaks. We call this simple methodology

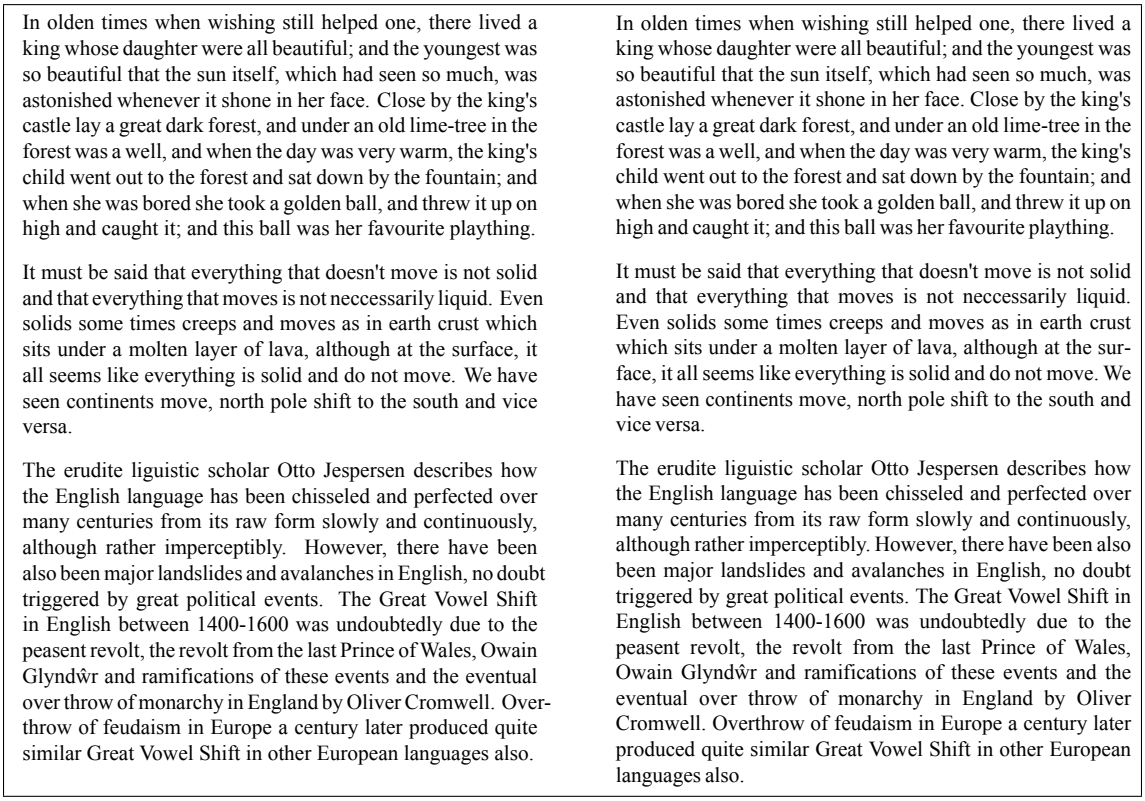


Figure 1: Rendering at 290pt text width; XeLaTeX on left, Firefox on right

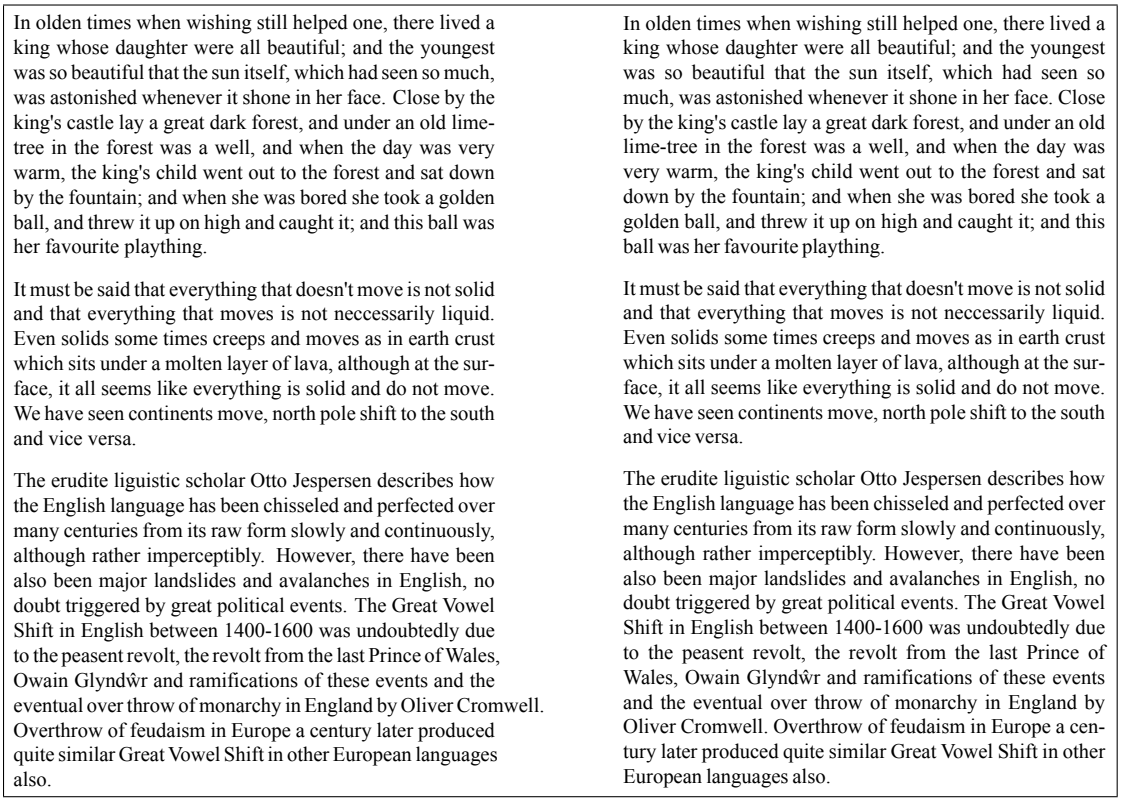


Figure 2: Rendering at 280pt text-width; XeLaTeX on left, Firefox on right.

the cookie-cutter algorithm and have applied for a patent, which is pending approval at the Indian patent office [4]. This algorithm has been tested for hundreds of sample articles and has been found to be quite successful at producing satisfactory paginated results. It is true that the results produced are sub-optimal but are sufficiently close to optimum to be of acceptable quality.

After the fundamental work on automatic pagination by Plass [5], which showed that the optimal solution to the float placement problem is NP-complete, this problem has been reconsidered in a different light by Bruggemann-Klein et al. [6]. Mittelbach [7] implemented a new algorithm in LaTeX 2ε and Marriott et al. [8] studied the problem for multicolumn layout. However, all these methods are quite complex and involve solving complex optimization problems using a dynamic programming approach. Here, our cookie-cutter method [4] is not trying to solve for a global optimum but instead implements a simple one-pass float placement algorithm that seems to produce reasonable results for a large number of documents.

8 Conclusions

TeX is now facing interesting challenges from the HTML5 browser world. TeX with its own inner beauty and typographic elegance can make the transformation from codex to the scroll page as clearly demonstrated by Boris Veytsman [3]. It now requires the creation of a modern TeX browser, probably with a Unicode model as in XeTeX or LuaTeX, not altogether different from the spirit of IBM Tech Explorer. Doug McKenna [9] has shown that it is possible to rewrite TeX in portable C using his JSBox library. Once we have the box-model output quite like the HTML DOM model, then it is possible to manipulate such box-models with a comprehensive browser in the spirit of HTML5. This will be another exciting browser that will be part of the future world of tablets and devices that can safely carry forward the tradition of TeX into the next few decades.

◇ S.K. Venkatesan
TNQ Books and Journals Pvt. Ltd.
Chennai 600033, India
skvenkat (at) tnq dot co dot in

References

[1] Donald Knuth (2010). An Earthshaking Announcement, *TUGboat* 31:2, 121–124. http://tug.org/TUGboat/tb31-2/tb98knut.pdf

[2] Theodor Holm Nelson, Project Xanadu. Xanalogical Structure, Needed Now More than Ever: Parallel Documents, Deep Links to Content, Deep Versioning and Deep Re-Use, Keio University. http://www.xanadu.com.au/ted/XUsurvey/xuDation.html

[3] Boris Veytsman and Michael Ware (2011). Ebooks and paper sizes: Output routines made easier, *TUGboat* 32:3, 261–265. http://tug.org/TUGboat/tb32-3/tb102veytsman-ebooks.pdf

[4] S.K. Venkatesan, M.V. Bhaskar, Srikanth Vittal (2015). Transformation Of Marked-up Content To A Reversible File Format For Automated Browser Based Pagination, Application number 3348/CHE/2015, Indian Patent Office.

[5] Michael Plass (1981). Optimal pagination techniques for automatic typesetting systems. PhD thesis, Stanford University.

[6] A. Bruggemann-Klein, R. Klein, and S. Wohlfeil (1995). Pagination reconsidered, *Electronic Publishing* 8:2–3, 139–152. http://cajun.cs.nott.ac.uk/compsci/epo/papers/volume8/issue2/2point9.pdf

[7] Frank Mittelbach (2000). Formatting documents with floats: A new algorithm for LaTeX 2ε, *TUGboat* 21:3, 278–290. http://tug.org/TUGboat/tb21-3/tb68mittel.pdf

[8] Kim Marriott, Peter Moulder and Nathan Hurst (2007). Automatic float placement in multi-column documents, DocEng'07: Proceedings of the 2007 ACM symposium on Document engineering, 125–134.

[9] Doug McKenna (2014). On tracing the trip test with JSBox, *TUGboat* 35:2, 157–167. http://tug.org/TUGboat/tb35-2/tb110mckenna.pdf

Recollections of a spurious space catcher

Enrico Gregorio

Abstract

Several pitfalls are waiting for us when programming in (LA)TEX. This paper will examine some and search for a solution. Shall we find the promised land? Maybe so, with `expl3`.

1 Introduction

Programming in (LA)TEX is not easy to begin with, mainly because of idiosyncrasies of the language, but also with some subtler points due to the simple fact that the language is oriented to *typeset* text. Knuth has done his best to ease typing a document without caring too much about white space: under normal circumstances, strings of spaces and tabulations are treated as a single space; line endings are converted into a single space, unless followed by another line ending with only white space intervening; white space at the beginning of lines is ignored; spaces are stripped off at the end of a line and substituted by an internal end-of-line marker.

The precise rules are explained in detail in *The TEXbook* [2] and *TEX by Topic* [1], among others, and this paper is not the place to go into the gory details. I'll return to some of the above points. Authors can insert unintentional spaces in text too, but reporting that is not the purpose here.

Some of the readers may know me by my activity on TeX.StackExchange (as user `egreg`) where, according to one of the most esteemed member of the community, I have gained the greater part of my reputation by catching spurious spaces in TEX code.[1] Nevertheless, catching spurious spaces in code is sometimes a tough task: they can be hidden in rather obscure corners and it may be necessary to resort to `\tracingall` or similar heavy machinery in order to isolate them: TEX macros call other macros, often in very involved ways.

Anyone who has undertaken the job of writing macros, be they simple or tremendously complicated, has been bitten by a spurious space left in the code. It happens! Reformatting code is perhaps one of the most frequent reasons: a line is too long and we want to improve code readability so we split it, forgetting the magical `%` at the end of the line.

Another relevant aspect of TEX's language is that under certain conditions spaces are *ignored*: they are there and TEX behaves as if they aren't, but with a very peculiar feature that will be described later.

We have to distinguish carefully between *typed spaces* and *space tokens*. Only typed spaces are subject to the above 'contraction rule', whereas *space tokens* are not. As a simple example, `\space\space` will produce two spaces: when TEX expands `\space`, it is converted to a space token.

The paper will describe the most common errors, with examples taken from questions in TeX.StackExchange or from package code. I'll give no precise references, because the purpose of the paper is not to shame anybody.[2] It will end with some considerations about methods for avoiding these quirks.

2 First examples

A very famous spaghetti western movie is "Il buono, il brutto e il cattivo" by Sergio Leone, featuring Clint Eastwood, Lee Van Cleef and Eli Wallach. The English title is "The Good, the Bad and the Ugly". I like to present examples in this form and I'll do it now.

2.1 The ugly

```
\providecommand{\sVert}[1][0]{
\ensuremath{\mathinner{
\ifthenelse{\equal{#1}{0}}{ % if
```

[1] We like to joke quite much in the site's chatroom. I'm not completely sure that that remark is a joke.
[2] Maybe I'll make a couple of exceptions.

```
\rvert}{}
\ifthenelse{\equal{#1}{1}}{ % if
\bigr\rvert}{}
\ifthenelse{\equal{#1}{2}}{ % if
\Bigr\rvert}{}
\ifthenelse{\equal{#1}{3}}{ % if
\biggr\rvert}{}
\ifthenelse{\equal{#1}{4}}{ % if
\Biggr\rvert}{}
}} % \ensuremath{\mathinner{
}
```

This code is part of a package in which no blank line separates the various macro definitions and is ugly in several respects: there is no indentation that can make clearer the various parts and the nesting of the conditionals; the code is clumsy and misses several end-of-line `%`-protections.

Remember that an end-of-line is converted to a space token; in this particular case they aren't really relevant, because most of the code is processed in math mode where space tokens are ignored. However, a user could type `\sVert` in text mode because of `\ensuremath`, ending with excess spacing around the bar: in this case the first end-of-line and the space between `}}` and `%` at the end are *not* ignored.

Using `\providecommand` is obviously wrong: users loading the package will expect the command `\sVert` do the advertised thing, not something else. A better definition would be

```
\newcommand*{\sVert}[1][0]{%
  \ifcase#1\relax
    \rvert % 0
  \or
    \bigr\rvert % 1
  \or
    \Bigr\rvert % 2
  \or
    \biggr\rvert % 3
  \or
    \Biggr\rvert % 4
  \fi
}
```

I also consider it bad programming style to place the `%` next to a control word, but it's just an opinion. Those `%` characters after `\rvert` are not necessary if no comment is used.

2.2 The bad

```
\def\@wrqbar#1{%
\ifnum\value{page}<10\def\X{\string\X}\else%
\ifnum\value{page}<100\def\X{\string\Y}\else%
\def\X{\string\Z}\fi\fi%
\def\F{\string\F}\def\E{\string\E}%
\stepcounter{arts}%
\iffootnote%
\edef\@tempa{\write\@barfile{\string%
\quellentry{#1\X{}{\thepage}}{\F{}{\thefootnote}}}}%
\else%
\edef\@tempa{\write\@barfile{\string%
\quellentry{#1\X{}{\thepage}}{\E{}{\thearts}}}}%
\fi%
\expandafter\endgroup\@tempa%
\if@nobreak \ifvmode\nobreak\fi\fi\@esphack}
```

Here we surely can't find spurious spaces. There are even too many % characters! The main problem here is that it's almost impossible to read the code.

2.3 The good

```
\def\deleterightmost#1{\edef#1{\expandafter\xyzzy#1\xyzzy}}
\long\def\xyzzy\\#1#2{\ifx#2\xyzzy\yzzyx
  \else\noexpand\\{#1}\fi\xyzzy#2}
\long\def\yzzyx#1\xyzzy\xyzzy{\fi}
```

This code by the Grand Wizard can be taken as a model of neatness. However, it's not really clear what it does: reading Appendix D of *The TEXbook* is necessary to understand it.

2.4 A surprise

```
{\tt A'C B}

\def\adef#1{\catcode`#1=13 \begingroup
 \lccode`\~=`#1\lowercase{\endgroup\def~}}
\let\oldtt\tt\def\tt{\adef'{\char"0D}\oldtt}

{\tt A'C B}

{\tt A'c B}
\bye
```

Run this code with plain TEX and you'll have quite a surprise. The purpose is to substitute the curly apostrophe with the straight one that the `cmtt10` font has in position `"0D`. The surprise is that we just get `AB` with no space in between, no quote and no `C`, but a mysterious warning in the log file:

```
Missing character: There is no ^^dc in font cmtt10!
```

This example has no spurious space; instead, it has a missing one! (Read on for details.)

2.5 Great programmers are not immune

This is an excerpt from `cleveref.sty`, a great piece of software nonetheless:

```
\cref@addlanguagedefs{spanish}{%
 \PackageInfo{cleveref}{loaded `spanish' language definitions}
 \renewcommand{\crefrangeconjunction}{ a\nobreakspace}%
 \renewcommand{\crefrangepreconjunction}{}%
 \renewcommand{\crefrangepostconjunction}{}%
 \renewcommand{\crefpairconjunction}{ y\nobreakspace}%
 [...]
```

Perhaps the author tested the language support only in some cases: the spurious space in the second line showed up when a user tried something like

```
The Spanish word for Spain is \foreignlanguage{spanish}{Espa\~na}
```

and realized that there were *two* spaces between 'is' and 'España'. Isolating the problem was far from easy, because cleveref wasn't seemingly involved. The macro `\cref@addlanguagedefs` adds to `\extrasspanish`, which is executed every time the language shifts to Spanish: so at this language change a space was produced.

See Figure 1 for another example.

3 What happens?

The rule is simple: an end-of-line is converted into a 'typed space' that *can* become a space token, but won't if it follows a control word. So, in the line

```
\iffootnote%
```

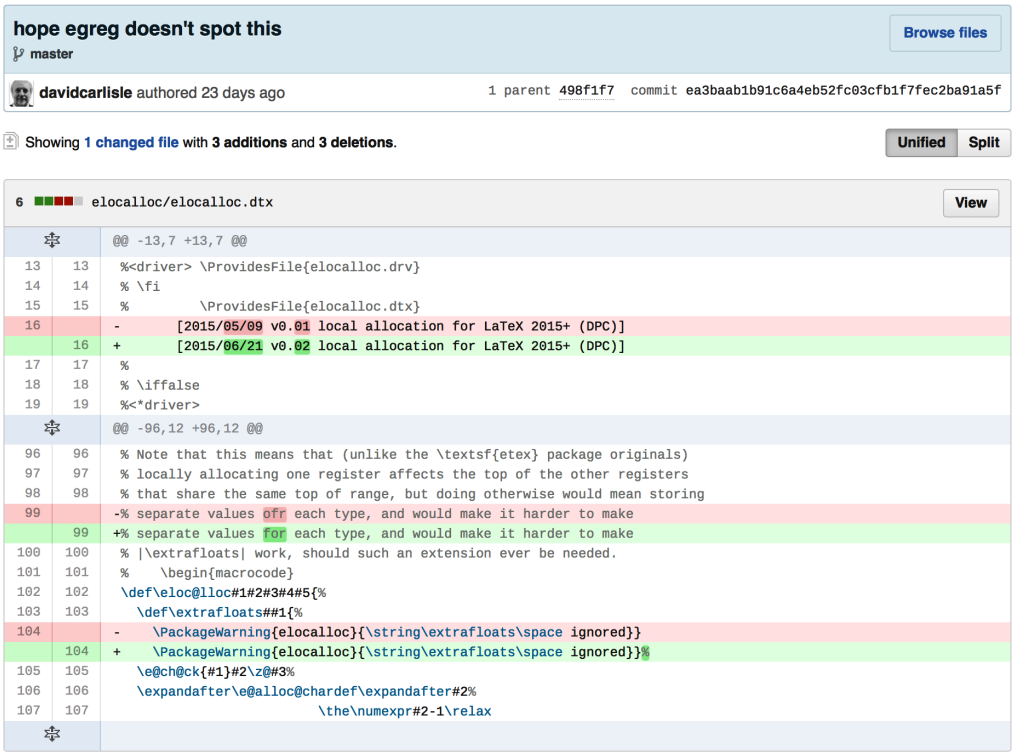


Figure 1: Another example of expert TEXnicians not being immune

the comment character is not necessary because the typed space resulting from the end-of-line will be ignored, being after a control word. To the contrary, the end-of-line in

```
\PackageInfo{cleveref}{loaded ‘spanish’ language definitions}
```

becomes a regular space token in the replacement text of the macro being defined and will disappear only if TEX is in a good mood when the macro is used.

Being in a good mood about space tokens means that the current mode is vertical (between paragraphs, basically) or math: in these cases, space tokens do nothing. If TEX is typesetting a regular paragraph (or a horizontal box), space tokens are generally *not* ignored and this is the cause for the mysterious effect presented in section 2.5.

Here are some examples of code written by people probably accustomed to other programming languages where spaces are used much more freely to separate tokens and are mostly irrelevant (unless in a string, of course).

```
\newcommand{\smallx}[1]{
     \begin{center}
        \begin{Verbatim}[commandchars=\\\{\}]

            \code{#1}

        \end{Verbatim}
    \end{center}
 }
============================
\include{fp}
\newcommand\entryOne[1]{
\ifnum #1 = 100 #1 \fi
```

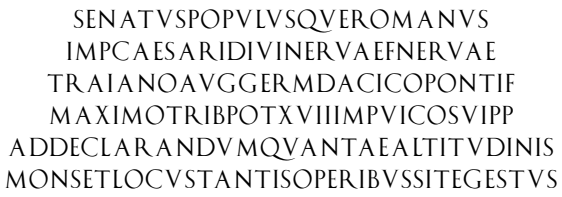


Figure 2: The inscription at the base of the *Columna Traiana* in Rome

```
}
\newcommand\entryTwo[1]{
\FPeval{\result}{#1}
\ifnum \result = 100 \result \fi
}
==========================
\newcommand{\trellis}[4]{
  \def \STATES {#1}
  \def \PSK {#2}
  \def \XDISTANCE {#3}
  \def \YDISTANCE {#4}
  \FPupn\NGROUPS{\STATES{} \PSK{} div 0 trunc}
  \multido{\ryA=0+-\YDISTANCE,\nA=1+1}{\STATES}{%
    \dotnode(0,\ryA){dotA\nA}
    \dotnode(\XDISTANCE,\ryA){dotB\nA}
  }
  \multido{\nG=1+1,\nOffset=1+\PSK}{\NGROUPS}{%
    \multido{\nStart=\nG+\NGROUPS}{\PSK}{%
      \multido{\nArrows=\nOffset+1}{\PSK}{%
        \ncline{dotA\nStart}{dotB\nArrows}
      }
    }
  }
}
```

The authors of the first two examples have no notion of spurious space and write TEX code as if it was, say, C. The third example mixes end-of-line protection with 'free form code'. Counting the number of spurious spaces so produced is a funny exercise.

Unfortunately, TEX is not free form! Spaces *are* important in typesetting, unless we want to write texts like the ancient Romans did: in Figure 2 we can see an emulation of the inscription at the base of the *Columna Traiana* in Rome.[3] (The font is Trajan by Peter Wilson, `http://ctan.org/pkg/trajan`.)

The ancients did not use spaces for several reasons. The suffixes helped in dividing a word from the next one, but perhaps the most important reason was saving space: marble and parchment were very expensive. Only the introduction of print and of less expensive paper allowed for using spaces between words for better clarity and ease of reading.

Well, one says, why don't we add `%` at the end of each line in macro code and forget about the whole thing, even if the code is a bit harder to read?

Sorry, no. Here's another example that shows this is not possible: there are two diseases that I call the 'spurious space syndrome' and the 'missing space syndrome'. The latter is the more serious one.

```
\documentclass{article}
\newcount\monthlycount
```

[3] When writing the paper, I was victim of a spurious space sneaking in the text of the inscription, but fortunately I noticed it before submission.

```
\newcommand{\monthlytodo}[1]{\par%
  \fbox{%
    \parbox{10cm}{%
      \monthlycount=1%
      \loop\ifnum\monthlycount<13%
        #1--\number\monthlycount\hrulefill\par%
        \advance\monthlycount by 1%
      \repeat%
    }%
  }%
}
\begin{document}
\monthlytodo{2013}
\end{document}
```

This code should print a boxed numbered lists, with items consisting of a rule preceded by year and month number. Running it will make TEX stop, after several seconds with no sign of activity, showing

```
! TeX capacity exceeded, sorry [main memory size=5000000].
<to be read again>
                   -
l.15 \monthlytodo{2013}
```

Quite surprising, isn't it? Well, the author of the code had in the past been a victim of the spurious space syndrome, so he started to add `%` at the end of each line, even in the document part, not only in macro code.

Let's see what happens: the replacement text of the macro would be shown by TEX as

```
... \loop \ifnum \monthlycount <13#1--\number \monthlycount ...
```

and, when `\monthlytodo{2013}` is called, the test will be

```
... \loop \ifnum \monthlycount <132013--\number \monthlycount ...
```

No wonder now that it takes so long to end the loop and that TEX runs out of memory, because it's trying to build an `\fbox`!

A clear case of 'missing space syndrome'. The solution is *not* having `%` after `13`. Since the constant is part of a numeric test, we are in the special case where TEX *ignores* a space token after it.

The code in section 2.4 suffers from the same syndrome; when `{\tt A'C B}` is processed, the apostrophe becomes active and expanded like a macro, leading to the token list

```
A\char"0DC B
```

and this is where things go wrong: `\char` looks for a number in hexadecimal format because of `"` and finds the digits `DC`; since no character lives in that slot, the error message about a missing character `^^dc` is issued. The correct code should have either a space or `\relax`

```
\let\oldtt\tt\def\tt{\adef'{\char"0D }\oldtt}
\let\oldtt\tt\def\tt{\adef'{\char"0D\relax}\oldtt}
```

Which one is a stylistic decision: both the space and `\relax` stop the search for further digits; `\relax` would do nothing, but the space would be ignored. There is another possibility, that is,

```
\let\oldtt\tt\def\tt{\adef'{\active}\oldtt}
```

but this exploits the incidental fact that `"0D` = 13 and that `\active` is defined with `\chardef` to point at character 13, so it's not good programming.

Here is a worse example discovered by Frank Mittelbach some days after the TUG meeting.[4]

```
2337              \advance\@tempcnta-2%
2338              \ifnum \thevpagerefnum =\@tempcnta%
```

[4] http://tex.stackexchange.com/questions/257100/varioref-and-previous-page

Remember the spurious spaces in `cleveref.sty` discussed in section 2.5? After I reported the bug, the author went on and added `%` at the end of lines. In this case it is a very bad case of 'missing space syndrome': when TEX is looking for a numerical constant, it looks for a space token after it or a token that can't be interpreted as a digit. In doing this it *expands* tokens, so it evaluates the conditional before having set the value of `\@tempcnta` because it still doesn't know whether the number is ended. The result is that the reference will not be correct. This code is part of a redefinition of a command in `varioref.sty` and, of course, the original code (pretty much identical otherwise) has no `%` after `-2`.

The precise rule is that when a syntactical construct allows for ⟨*one optional space*⟩, TEX will look for it with expansion. Quoting *The TEXbook* [2, p. 208]

> For best results, always put a blank space after a numeric constant; this blank space tells TEX that the constant is complete, and such a space will never "get through" to the output. In fact, when you don't have a blank space after a constant, TEX actually has to do more work, because each constant continues until a non-digit has been read; if this non-digit is not a space, TEX takes the token you did have and backs it up, ready to be read again.

One could maintain that a macro writer using `\loop` should know the details of TEX programming, as this macro is not officially supported by LATEX, so the memory exhaustion problem would not show up. However, a gross estimate of the number of packages using `\loop` is 378. I didn't even try checking how many use `\ifnum`: both are essential devices in the (LA)TEX programmer's toolbox.

Surely a wannabe macro writer should be aware of these issues, but they are very subtle and, as we saw, even very experienced programmers can be victim of the bad syndrome. I could give plenty of similar examples.

4 Solutions

A possible way out of the space related syndromes might be fencing macro code with statements equivalent to doing

```
\catcode‘ =9 \endlinechar=-1 \makeatletter
```

(the last command for LATEX management of internal macros) with an appropriate restore action at the end. Something like

```
\edef\restorecodes{%
  \catcode32=\the\catcode32
  \endlinechar=\the\endlinechar
  \noexpand\makeatother
}
\catcode32=9 \endlinechar=-1 \makeatletter

... macro code ...

\restorecodes
```

but this wouldn't cure the missing space syndrome, but rather aggravate it because we can no longer add the optional space after constants! Moreover, sometimes we need space tokens in macro code. Well, we could add `\space` where we want a space, but this is cumbersome. Another bright idea comes to mind: use `~` as space. Let's try it.

```
\edef\restorecodes{%
  \catcode32=\the\catcode32
  \catcode126=\the\catcode126
  \endlinechar=\the\endlinechar
  \noexpand\makeatother
}
\catcode32=9 \catcode126=10 \endlinechar=-1\makeatletter

\newcount\monthlycount
```

```
\newcommand{\monthlytodo}[1]{\par
  \fbox{
    \parbox{10cm}{
      \monthlycount=1~
      \loop\ifnum\monthlycount<13~
        #1--\number\monthlycount\hrulefill\par
        \advance\monthlycount by 1~
      \repeat
    }
  }
}

\restorecodes
```

This is indeed more like free form. Recall that category code 9 means 'ignored' and category code 10 means 'space'. But wait a moment! The rule says that spaces are stripped at end of lines and substituted with the internal end-of-line marker, in this case nothing because the parameter `\endlinechar` is set to −1. And maybe we wouldn't have 'real spaces' in the replacement text of our macros. Again a couple of quotations from *The TEXbook* solve the issue. First about the stripping of spaces [2, p. 46]

> TEX deletes any ⟨*space*⟩ characters (number 32) that occur at the right end of an input line. Then it inserts a ⟨*return*⟩ character (number 13) at the right end of the line [...]

and this is supplemented by the fact that `\endlinechar` usually has the value 13. So the category code 10 tilde will not be stripped, because its character code is not 32.[5]

About the second issue, look at [2, p. 47]

> If TEX sees a character of category 10 (space), the action depends on the current state. If TEX is in state *N* or *S*, the character is simply passed by, and TEX remains in the same state. Otherwise TEX is in state *M*; the character is converted to a token of category 10 whose character code is 32, and TEX enters state *S*. The character code in a space token is always 32.

Thus we can count on `~` being converted to a real space token when the replacement text of the macro is stored in memory. Of course we lose the character to denote a tie, but when writing macros this is rarely needed and we can always resort to `\nobreakspace` in those cases.

Why not use `\relax` instead? It *can* be used, of course, but we lose full expandability that, in several cases, is what we need. Note that integer registers or constants defined with `\chardef` or `\mathchardef` are already 'full' numbers on their own and no space is looked for after them.

5 LaTeX3 and expl3, a new way around these problems and much more

Whoever is so kind to follow my answers at TeX.StackExchange will have probably understood where I'm going: the LaTeX3 project started more than twenty years ago, but has been stalling for a long time until a few years ago, when it became really possible to use the new programming paradigms it introduced. At the time the LaTeX team started studying it, computing resources were too scant: for instance, running LaTeX 2ε on emTEX left very little space for labels and personal macros. Nowadays, we can make presentations, plots, complex drawings with very short computing time. I remember without any nostalgia the times when drawing a mildly complicated diagram using PICTEX could take minutes! Making a 37 frame presentation from this paper just took seconds, and it involved compiling twice with a run of PythonTEX in between.

Thus the overhead of loading several thousand lines of code is not much of a problem as it could have been years ago. When these lines of code are included in the format, the loading time will be reduced to a few milliseconds.

[5] It must be mentioned that the current TEX implementations of TEX Live and MiKTEX also strip off tabs (character code 9).

The expl3 programming environment provides 'code block fences' similar to the ones I described earlier, but it also adds _ and : as characters that can be used in control sequence names. Think of _ as the traditional @ (that's not allowed in control sequence names); the use of the colon is rather interesting, and I'll return to it after having given some examples.

I'm aware that changing one's programming paradigm can be difficult at first, because habits are hard to die. But a couple of toy problems may be able to get your attention.

First problem: we want to obtain the ratio between two lengths in an expandable way to use, for instance, as a factor in front of another length parameter and we want this with as high accuracy as possible.

```
\documentclass{article}
\usepackage{xparse}

\ExplSyntaxOn % start the programming environment
\DeclareExpandableDocumentCommand{\dimratio}{ O{5} m m }
 {
  \fp_eval:n
   {
    round ( \dim_to_fp:n { #2 } / \dim_to_fp:n { #3 } , #1 )
   }
 }

\ExplSyntaxOff % end the programming environment
```

Apart from line breaks added for readability, this is essentially one line of code! And it can be 'free form'! I use the \fp_eval:n function that produces the result of a computation, together with data type conversion functions. If we try \dimratio{\textwidth}{\textheight} or \dimratio[2]{\textwidth}{\textheight} we obtain, respectively, 0.62727 and 0.63 and we could even say

```
\setlength{\mylength}{\dimratio{\textwidth}{\textheight}\mylength}
```

in order to scale the value of the parameter \mylength by this ratio.

Second toy problem. We want to parse a semicolon-separated list, applying to each item some formatting macro \dosomething. First some nice code by Petr Olšák:

```
\def\ls#1{\lsA#1;;}
\def\lsA#1;{\ifx;#1;\else \dosomething{#1}\expandafter\lsA\fi}
\def\dosomething#1{\message{I am doing something with #1}}

\ls{(a,b);(c,d);(e,f)}
```

This is a well-known technique which does its job nicely, but has some shortcomings that I'll illustrate after showing the corresponding expl3 code:

```
\ExplSyntaxOn
\NewDocumentCommand{\dosomething}{m}
 {
  I ~ am ~ doing ~ something ~ with ~ #1
 }
\seq_new:N \l_manual_ls_items_seq
\NewDocumentCommand{\ls}{m}
 {
  \seq_set_split:Nnn \l_manual_ls_items_seq { ; } { #1 }
  \seq_map_function:NN \l_manual_ls_items_seq \dosomething
 }
\ExplSyntaxOff

\ls{(a,b); (c,d) ;(e,f)}
```

One of the new data types introduced in `expl3` is the *sequence* type: an ordered list of items, which can be accessed as a whole or by item number. The first function does the splitting and loads the declared variable with the items; then `\seq_map_function:NN` passes each item as the argument to the function `\dosomething`, which is just the same as Petr Olšák's macros.

Oh, wait! If you look carefully, in my example there are spaces surrounding the middle item, which would sneak into Petr's code, while they don't with the `expl3` code, because the splitting function automatically trims off leading and trailing spaces around an item. So users can even type the input as

```
\ls{
  (a,b);
  (c,d);
  (e,f)
}
```

if they deem it convenient.

Let's go back to the Grand Wizard's code shown in section 2.3:

```
\def\deleterightmost#1{\edef#1{\expandafter\xyzzy#1\xyzzy}}
\long\def\xyzzy\\#1#2{\ifx#2\xyzzy\yzzyx
  \else\noexpand\\{#1}\fi\xyzzy#2}
\long\def\yzzyx#1\xyzzy\xyzzy{\fi}
```

It's supposed to remove the last item from a sequence. In Appendix D [2, p. 378], sequences, called *list macros*, are implemented as macros with a replacement text such as `\\{(a,b)}\\{(c,d)}\\{(e,f)}`, so one could do

```
\def\myitems{\\{(a,b)}\\{(c,d)}\\{(e,f)}}
```

and the `\deleterightmost` macro is supposed to be called like

```
\deleterightmost\myitems
```

in order to remove the last item, so leaving the same as if the definition had been

```
\def\myitems{\\{(a,b)}\\{(c,d)}}
```

Compare the code above with the `expl3` code

```
\seq_pop_right:NN \l_manual_ls_items_seq \l_tmpa_tl
```

which has also the significant advantage that the last item is still available in the token list scratch variable `\l_tmpa_tl` (or any token list variable we choose to use). Knuth's macro simply discards it.

Here's a new version for the `\monthlytodo` macro:

```
\documentclass{article}
\usepackage{xparse}

\ExplSyntaxOn
\NewDocumentCommand{\monthlytodo}{ O{10cm} m }
 {
  \par \noindent \fbox { \manual_monthlytodo:nn { #1 } { #2 } }
 }
\cs_new_protected:Nn \manual_monthlytodo:nn
 {
  \parbox { \dim_eval:n { #1 - 2\fboxsep - 2\fboxrule } }
   {
    \int_step_inline:nnnn { 1 } { 1 } { 12 }
     {
      #2--\int_to_arabic:n { ##1 }\hrulefill\par
     }
   }
 }
```

```
\ExplSyntaxOff
\begin{document}
\monthlytodo{2013}
\monthlytodo[\textwidth]{2015}
\end{document}
```

Note that there's no `\loop` any more, but the much more convenient `\int_step_inline:nnnn` function that accepts as its first three arguments, respectively, the starting point, the step and the ending point; the fourth argument contains code that should use the current value, available as `#1` which in this case must become `##1` because we're doing a definition. I also added an optional argument to set the width of the box and show `\dim_eval:n`.

No need to be obsessed by adding spaces after constants, because objects are clearly separated from each other. If a function needs a numeric argument, it will be given in braces and the function's *signature* tells us how many arguments are expected (of course, one has to know what type of argument should be given).

Just to show the power of `expl3`, let me make a bimonthly calendar:

```
\cs_new_protected:Nn \manual_bimonthlytodo:nn
 {
  \parbox { \dim_eval:n { #1 - 2\fboxsep - 2\fboxrule } }
   {
    \int_step_inline:nnnn { 1 } { 2 } { 12 }
     {
      #2 --
      (\int_to_arabic:n { ##1 } -- \int_to_arabic:n { ##1 + 1 })
      \hrulefill\par
     }
   }
 }
```

where the step is two; only six steps will be performed, because at the next the value would be above the stated upper bound: let TEX do the computations.

In `expl3` a careful distinction is made between *functions* and *variables*. They are of course all realized as macros or registers; the distinction is in what they are for: functions do something, whereas variables store tokens.

The naming scheme makes it easy to distinguish between them: a function has a signature made of zero or more characters, separated from the name by a colon. The introductory manual [7] explains what characters are valid and what they mean. The name of a variable should start with `l_`, `g_` or `c_`, meaning *local*, *global* or *constant*; `expl3` provides distinct functions to act on local and global variables; constants should just be allocated and given a value. A common source of head scratching is memory exhaustion due to filling up the save stack; always adding in the correct way to variables should avoid the problem. The interface manual [6] describes all available kernel functions; there are also other manuals for the still-experimental packages such as the one for regular expressions [8], and not to forget the higher level command defining package xparse [9].

6 Advantages and disadvantages

Powerful tools always have pros and cons. Let me make a short list of the good parts I've found in `expl3`.

1. Consistent interface: the team is striving for a set of basic functions in such a way that the name suggests the action.
2. Several new data types: TEX basically has only macros, but `expl3` builds higher level structures that help in keeping different things apart.
3. Tons of predefined functions: the most common tasks when operating on data structures are provided and the team is usually responsive when uncovered use cases are shown.

4. Ongoing development: the state of `expl3` is quite stable; the good thing is that it hides the implementation details of data types and basic functions, so changes at the lower level will have no effect on higher level constructs (apart from speed and efficiency).

   Here's a list of the new data types:

token lists are generic containers for tokens (think of `\chaptername`);

sequences are ordered sets of token lists;

comma separated lists are the same, more user interface oriented;

property lists are unordered sets of token lists, addressed by key;

floating point numbers adhere to the IEEE standard;

regular expressions are as near to the POSIX standard as possible;

coffins are ordinary TEX boxes but with many more handles.

The available data types of course include booleans, integers, boxes, lengths, skips (rubber lengths), input and output streams as in standard TEX.

Several papers in *TUGboat* have touched on the problem of case switching macros; here's a simple example of how this is done in `expl3`:

```
\str_case:nnTF { <string> }
 {
  {a}{Case~a}
  {b}{Case~b}
  {z}{Case~z}
 }
 {Additional code if there is a match}
 {Code if there is no match}
```

The ⟨*string*⟩ mentioned in the first argument is usually the argument to a function/macro. This also shows the purpose of function signatures: the function expects four braced arguments, the last two denote code to execute if the test returns true (in this case that the first argument is found in the list specified in the second argument) or false, respectively. Also `\str_case:nn`, `\str_case:nnT` and `\str_case:nnF` are provided, for cases when the ⟨*false code*⟩ or ⟨*true code*⟩ are not needed, respectively: an example of what I mean by 'consistent interface'.

Another example of consistent interface is the following. Suppose we have to do something with two token lists, one of which is explicitly given and the other one is sometimes only available as the contents of a token list variable.

```
\cs_new_protected:Nn \manual_foo:nn
 {
  Do~something~with~#1~and~#2
 }
\cs_generate_variant:Nn \manual_foo:nn { nV , Vn , VV }

\manual_foo:nn { Bar } { Foo }
\manual_foo:nV { Bar } \l_manual_whatever_tl
\manual_foo:Vn \l_manual_whatever_tl { Foo }
\manual_foo:VV \l_manual_whatever_a_tl \l_manual_whatever_b_tl
```

The *variant* is defined in a consistent way, giving us four similarly named functions that perform the same action, only on differently specified arguments. Doing the same in standard LATEX requires juggling arguments and well placed `\expandafter` tokens: a nice game to play, but usually leading to undesired code duplication. The `V` type argument means that the argument is expected to be a variable, whose value is obtained and placed in a braced argument for processing with the base function.

Now let's turn to the cons. The code is much more verbose, rather like C code is much more verbose than assembler; a price to pay when the pool of available base function is larger. One has never (well, almost) to use `\expandafter`, the preferred toy of all TEX programmers;

in particular, no `\expandafter\@firstofone`. (Er, just joking.) Coding in `expl3` still requires understanding how macro expansion works and error messages can be quite cryptic, as they often refer to TEX's lowest level; however, also programming in plain TEX can lead to inscrutable errors, so this is already a problem.

7 The real advantages

The `.../tex/latex` subtree in TEX Live contains more than 2000 directories. If we look at those packages, we can easily see that several of them reinvent the wheel many times. It's not rare to even see LATEX kernel provided functions/macros reimplemented! The `\ls` macro described above, with its auxiliary macro `\lsA` is an example: the code is good, but it can be found umpteen times in small variations, for every parsing task. The same can be said for dozens of typical constructs, so having a wide base of functions for common tasks is going to make code much more readable and understandable. Studying the `\xyzzy` example is certainly fun and instructive for grasping concepts related to recursive macros, particularly tail recursion. But why redo the work the LATEX team has already done for us?

Frank Mittelbach's papers (with Rainer Schöpf and Chris Rowley) [3, 4, 5] about LATEX3 are very interesting reading for understanding the aim of the project.

So it's not just about 'avoiding spurious spaces' or not 'forgetting required spaces': we have available, and in a quite mature state, an almost *complete* (and extendable) programming environment that frees us from thinking about the low-level stuff, while concentrating on the real programming task.[6]

Since the meeting has taken place in Darmstadt, Germany, I'd like to finish in the German language, with apologies to David Hilbert:

Aus dem Paradies,
das das LATEX3 Team uns geschaffen,
soll uns niemand vertreiben können.

(From the paradise that the LATEX team created for us, no one can expel us.)

References

[1] Victor Eijkhout. *TEX by Topic, A TEXnician's Reference*. Addison-Wesley, Reading, MA, USA, 1992. `http://eijkhout.net/texbytopic/`.

[2] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[3] Frank Mittelbach and Chris Rowley. LATEX 2.09 ↪ LATEX3. *TUGboat*, 13(1):96–101, April 1992. `http://tug.org/TUGboat/tb13-1/tb34mittl3.pdf`.

[4] Frank Mittelbach and Chris Rowley. The LATEX3 Project. *TUGboat*, 18(3):195–198, September 1997. `http://tug.org/TUGboat/tb18-3/l3project.pdf`.

[5] Frank Mittelbach and Rainer Schöpf. Towards LATEX 3.0. *TUGboat*, 12(1):74–79, March 1991. `http://tug.org/TUGboat/tb12-1/tb31mitt.pdf`.

[6] The LATEX Project. The LATEX3 interfaces, 2015. `http://mirror.ctan.org/macros/latex/contrib/l3kernel/interface3.pdf`.

[7] The LATEX Project. The expl3 package and LATEX3 programming, 2015. `http://ctan.org/pkg/l3kernel`.

[8] The LATEX Project. The l3regex package: Regular expressions in TEX, 2015. `http://ctan.org/pkg/l3regex`.

[9] The LATEX Project. The xparse package: Document command parser, 2015. `http://ctan.org/pkg/xparse`.

◇ Enrico Gregorio
Università di Verona, Dipartimento di Informatica
Strada le Grazie 15, Verona, Italy
`enrico dot gregorio (at) gmail dot com`

[6] In a conversation at the TUG meeting, Stefan Kottwitz asked me to remark on the best advancements with `expl3`; I answered 'floating point computations' and 'regular expressions'. These are of course implemented with TEX primitives, but my opinion is that they wouldn't be so powerful if `expl3` had not been developed.

When to stop ...

Hans Hagen

Abstract

A flexible system like TEX permits all kinds of solutions for (weird) problems, so as soon as you run into a special case it is tempting to come up with a solution. When many such solutions are built into a macro package at some point they start to compete. How far should one go in being nice for users and customers, especially when demands are mostly based on tradition, expectations, and of course subjective, non-rational and disputable artistic considerations? This article looks at three examples: removing already-typeset material, support for the ASCIIMATH format, and profiling. Do we really need these, and if so, where and when do they need to be available?

1 Removing material already typeset

The primitive `\unskip` often comes in handy when you want to remove a space (or more precisely: a glue item) but sometimes you want to remove more. Consider for instance the case where a sentence is built up stepwise from data. At some point you need to insert some punctuation but as you cannot look ahead it needs to be delayed. Keeping track of accumulated content is no fun, and a quick and dirty solution is to just inject it and remove it when needed. One way to achieve this is to wrap this optional content in a box with special dimensions. Just before the next snippet is injected we can look back for that box (that can then be recognized by those special dimensions) and either remove it or unbox it back into the stream.

To be honest, one seldom needs this feature. In fact I never needed it until Alan Braslau and I were messing around with (indeed messy) bibliographic rendering and we thought it would be handy to have a helper that could remove punctuation. Think of situations like this:

```
John Foo, Mary Bar and others.
John Foo, Mary Bar, and others.
```

One can imagine this list to be constructed programmatically, in which case the comma before the `and` can be superfluous. So, the `and others` can be done like this:

```
\def\InjectOthers
  {\removeunwantedspaces
   \removepunctuation
   \space and others}
John Foo, Mary Bar, \InjectOthers.
```

Notice that we first remove spaces. This will give:

John Foo, Mary Bar and others.

where the commas after the names are coming from some not-too-clever automation or are the side effect of lazy programming. In the sections below I will describe a bit more generic mechanism and also present a solution for non-ConTEXt users.

1.1 Marked content

The example above can be rewritten in a more general way. We define a couple macros (using ConTEXt functionality):

```
\def\InjectComma
  {\markcontent
     [punctuation]
     {\removeunwantedspaces,\space}}
\def\InjectOthers
  {\removemarkedcontent[punctuation]%
   \space and others}
```

These can be used as:

```
John Foo\InjectComma
Mary Bar\InjectComma
\InjectOthers.
```

Which gives us:

John Foo, Mary Bar and others.

Normally one doesn't need this kind of magic for lists because the length of the list is known and injection can be done using the index in the list. Here is a more practical example:

```
\def\SomeTitle {Just a title}
\def\SomeAuthor{Just an author}
\def\SomeYear  {2015}
```

We paste the three snippets together:

```
\SomeTitle,\space \SomeAuthor\space (\SomeYear).
```

But to get even more abstract, we can do this:

```
\def\PlaceTitle
  {\SomeTitle
   \markcontent[punctuation]{.}}
\def\PlaceAuthor
  {\removemarkedcontent[punctuation]%
   \markcontent[punctuation]{,\space}%
   \SomeAuthor
   \markcontent[punctuation]{,\space}}
\def\PlaceYear
  {\removemarkedcontent[punctuation]%
   \space(\SomeYear)%
   \markcontent[punctuation]{.}}
```

Used as:

```
\PlaceTitle\PlaceAuthor\PlaceYear
```

we get the output:

Just a title, Just an author (2015).

but when we have no author:

```
\def\SomeAuthor{}
\PlaceTitle\PlaceAuthor\PlaceYear
```

now we get:

Just a title (2015).

Even more clever:

```
\def\SomeAuthor{}
\def\SomeYear{}
\def\SomePeriod
  {\removemarkedcontent[punctuation].}
\PlaceTitle\PlaceAuthor\PlaceYear\SomePeriod
```

The output is:

Just a title.

Of course we can just test for a variable like `\SomeAuthor` being empty before we place punctuation, but there are cases where a period becomes a comma or a comma becomes a semicolon. Especially with bibliographies your worst typographical nightmares come true, so it is handy to have such a mechanism available when it's needed.

1.2 A plain solution

For users of LuaTEX who don't want to use ConTEXt I will now present an alternative implementation. Of course more clever variants are possible but the principle remains. The trick is simple enough to show here as an example of Lua coding as it doesn't need much help from the infrastructure that the macro package provides. The only pitfall is the used signal (attribute number) but you can set another one if needed. We use the `gadgets` namespace to isolate the code.

```
\directlua {
  gadgets         = gadgets or { }
  local marking   = { }
  gadgets.marking = marking

  local marksignal   = 5001
  local lastmarked   = 0
  local marked       = { }
  local local_par    = 6
  local whatsit_node = 8

  function marking.setsignal(n)
    marksignal = tonumber(n) or marksignal
  end

  function marking.mark(str)
    local currentmarked = marked[str]
    if not currentmarked then
      lastmarked    = lastmarked + 1
      currentmarked = lastmarked
      marked[str]   = currentmarked
    end
    tex.setattribute(marksignal,currentmarked)
  end

  function marking.remove(str)
    local attr = marked[str]
    if not attr then
      return
    end
    local list = tex.nest[tex.nest.ptr]
    if list then
      local head = list.head
      local tail = list.tail
      local last = tail
      if last[marksignal] == attr then
        local first = last
        while true do
          local prev = first.prev
          if not prev
             or prev[marksignal] ~= attr
             or (prev.id == whatsit_node and
                 prev.subtype == local_par) then
             break
          else
            first = prev
          end
        end
        if first == head then
          list.head = nil
          list.tail = nil
        else
          local prev = first.prev
          list.tail  = prev
          prev.next  = nil
        end
        node.flush_list(first)
      end
    end
  end
}
```

These functions are called from macros. We use symbolic names for the marked snippets. We could have used numbers but meaningful tags can be supported with negligible overhead. The remover starts at the end of the current list and goes backwards till no matching attribute value is seen. When a valid range is found it gets removed.

```
\def\setmarksignal#1%
  {\directlua{gadgets.marking.
                setsignal(\number#1)}}

\def\marksomething#1#2%
  {{\directlua{gadgets.marking.mark("#1")}{#2}}}

\def\unsomething#1%
  {\directlua{gadgets.marking.remove("#1")}}
```

The working of these macros can best be shown from a few examples:

```
before\marksomething{gone}{\em HERE}%
  \unsomething{gone}after
before\marksomething{kept}{\em HERE}%
```

```
  \unsomething{gone}after
\marksomething{gone}{\em HERE}%
  \unsomething{gone}last
\marksomething{kept}{\em HERE}%
  \unsomething{gone}last
```

This renders as:

beforeafter
beforeHEREafter
last
HERElast

The remover needs to look at the beginning of a paragraph marked by a local par whatsit. If we removed that, LuaTEX would crash because the list head (currently) cannot be set to nil. This is no big deal because this macro is not meant to clean up across paragraphs.

A close look at the definition of `\marksomething` will reveal an extra grouping in the definition. This is needed to make content that uses `\aftergroup` trickery work correctly. Here is another example:

```
\def\SnippetOne
  {first\marksomething{punctuation}{, }}
\def\SnippetTwo
  {second\marksomething{punctuation}{, }}
\def\SnippetThree
  {\unsomething{punctuation} and third.}
```

We can paste these snippets together and make the last one use `and` instead of a comma.

```
\SnippetOne \SnippetTwo  \SnippetThree\par
\SnippetOne \SnippetThree\par
```

We get:

first, second and third.
first and third.

Of course in practice one probably knows how many snippets there are and using a counter to keep track of the state is more efficient than first typesetting something and removing it afterwards. But still it looks like a cool feature and it can come in handy at some point, as with the title-author-year example given before.

The plain code shown here is in the distribution in the file `luatex-gadgets` and gets preloaded in the `luatex-plain` format.

2 Supporting ASCIIMATH

In ConTEXt we have supported MathML for a long time alongside TEX input and we use it in projects mostly related to educational typesetting. In these days of multiple output from one source that sounds handy but unfortunately support in browsers is less than optimal, especially if you consider the long time MathML has been around. For that reason, in a recent project, the web folks involved in the project forced a move to something called ASCIIMATH. As with most ASCII-based encodings such a format starts out simple but due to demands it eventually becomes as complex as anything else.[1] In being tolerant to user input interesting side effects occur. You can imagine that when dealing with tens of thousands of files coming from dozens of authors keeping consistency is an issue. In spite of providing features for quality control (not hard in a TEX-driven backend) one runs into interesting situations.

$$o \otimes x = xo$$
$$aaxx = xa$$
$$\infty\times = xo$$
$$aa\times = xa$$

Figure 1: Surprising output.

$$ac + \sin x + x\sqrt{x} + \sin\sqrt{x} + \sin\sqrt{x}$$

Figure 2: Tolerance for space-less input.

The ASCIIMATH module is loaded with:

```
\usemodule[asciimath]
```

Here is an example of code:

```
\asciimath{o ox x = xo}
\asciimath{a ax x = xa}
\asciimath{ooxx=xo}
\asciimath{aaxx=xa}
```

This code produces the output in figure 1. Of course the expansion of some character sequences is officially defined but no one will memorize all of them.

Here is another input sequence:

```
\asciimath{ac+sinx+xsqrtx+sinsqrtx+sinsqrt(x)}
```

In figure 2 we see the result. The problem with this kind of input is that once spaces are that optional one loses a path to upward compatibility, and math is not a frozen language, so ...

A parser of ASCIIMATH is supposed to deal with numbers properly and as such, on the one hand assumes some syntactic consistent input, but at the same time needs to be tolerant to whatever input the author comes up with.

```
\asciimath{sqrt(1234)}
\asciimath{sqrt(1.234)}
\asciimath{sqrt(1,234)}
\asciimath{sqrt 1234}
\asciimath{sqrt 1.234}
\asciimath{sqrt 1,234}
```

The result of this input is shown in figure 3. As you can see here, only periods can be part of

[1] Long ago we started supporting something called calculator math because that was handy when discussing the use of calculators in school books.

$$\sqrt{1234}\ \sqrt{1.234}\ \sqrt{1,234}$$
$$\sqrt{1234}\ \sqrt{1.234}\ \sqrt{1,234}$$

Figure 3: Numbers as a unit of input.

$$\sqrt{1234567.1234567}$$
$$\sqrt{1234567,1234567}$$
$$\sqrt{1234567.1234567}$$
$$\sqrt{1234567},1234567$$

Figure 4: Sloppy input resulting in errors.

a number, a comma terminates one. But in some European countries commas are used instead; sadly, such cultural properties are seldom supported in these international times. When parentheses are used this gets unnoticed. One can argue that the spacing after a comma can be a signal to use parentheses (as argument delimiters). But, in an automated flow where the end users don't know much about how something math should be typeset you should not be too optimistic about that detailed of quality control.[2]

Of course some errors will get noticed, as in the following example, typeset in figure 4: a space is added after a comma (which is a punctuation character while a period is an ordinary character) and because a comma is not part of a number the root symbol is too low.

```
\asciimath{sqrt(1234567.1234567)}
\asciimath{sqrt(1234567,1234567)}
\asciimath{sqrt 1234567.1234567}
\asciimath{sqrt 1234567,1234567}
```

The use of `()` around the number will only obscure the error in coding the number. At some point the designers of the abovementioned (educational) math books decided that numbers should be split in triplets. Like this:

```
\setupasciimath
  [splitmethod=3,
   symbol={{,}}]
\asciimath{sqrt(1234567.1234567)}
\asciimath{sqrt(1234567,1234567)}
\asciimath{sqrt 1234567,1234567}
\asciimath{sqrt 1234567.1234567}
```

In figure 5 you see the result. This will only work well when periods are used in the input because a comma will end the number. In that case the spaces come out different too. So in order for that to work one really needs to input using periods. Those periods will be replaced by commas when typesetting.

Of course using commas in the input is a bad idea anyway as they can also separate coordinates

[2] The Dutch language market is relatively small, so there are limits to how much time and money publishers will spend.

$$\sqrt{1\ 234\ 567{,}123\ 456\ 7}$$
$$\sqrt{1\ 234\ 567}{,}1\ 234\ 567$$
$$\sqrt{1\ 234\ 567}{,}1\ 234\ 567$$
$$\sqrt{1\ 234\ 567{,}123\ 456\ 7}$$

Figure 5: Numbers grouped in triplets.

and in ASCIIMATH they are also used as separators between matrix entries. In the end, only very consistent and redundant coding will come close to guaranteeing a decent result. Unfortunately most input was not using periods, and that's kind of painful when it gets noticed at the last minute, especially when the demand for spaced numbers happens at the last minute too.

When you code in TEX directly there is normally more control over matters as there is a more direct relationship between authoring and rendering. For instance if you tag numbers like this, periods and commas are treated alike:

```
$\mn{1.2} = \mn{1,2}$
```

Here the `\mn` command reflects the MathML tag for numbers. To get this automatically you can say:

```
\setupmathematics
  [autopunctuation=yes]
```

In which case a comma is punctuation only if it's followed by a blank space.

The project where this ASCIIMATH is needed started over a decade ago[3] with TEX input but because the web was a target too we switched to content MathML. That not being supported, after a short excursion to OpenMath, we ended up with presentational MathML, and finally ASCIIMATH. The question here is not so much where to stop, but when can I stop adding more and more input methods. Quality is not well-served with ever-increasing variety and input tolerance. Unfortunately the choices are often determined by external factors. Interesting is that ConTEXt can produce MathML as a by-product so using TEX input works out fine.

One can of course ask in general when to stop with adding features. As we used some roots in the examples, here is another one (output in figure 6):

```
\setupmathradical
  [sqrt]
  [alternative=mp]

\sqrt{k\over m}
\sqrt{\displaystyle{k\over m}}
```

We mentioned the comma as cultural aspect of rendering numbers. I consider the small hook at the (right-hand) end of the root symbol another such —

[3] It concerns a free math methodology.

$$\sqrt{\frac{k}{m}} \quad \sqrt{\frac{k}{m}}$$

Figure 6: Another view of roots.

at least that's what got drawn on the blackboard when I attended math classes. We can provide this kind of rendering out of the box using MetaPost and it has neither a performance hit nor burdens the user. Unfortunately no one ever asked for this, while it is the kind of extension that I'm more than happy to provide. In my opinion it fits well with Don Knuth's "fine points of math typesetting" too. So I won't stop implementing such features.

3 Profiling lines

Although TEX is pretty good at typesetting simple texts like novels, in practice it's often used for getting more complex stuff on paper (or screen). Math is of course the first thing that comes to mind. If for instance you look at the books typeset by Don Knuth you will see a rendering that is rather consistent in spacing. This is no surprise as the author pays a lot of attention to detail and uses inline versus display math properly. No publisher will complain about the result.

In the documents that I have to write styles for, the content is rather mixed, and in particular inline math can have display math properties. In a one-column layout this is not a real problem especially because lots of short sentences and white space is used: we're talking of secondary-school educational math where arguments for formatting something this or that way is not always rational and consistent but more based on "this is what the student expects", "the competitor also does it that way" or just "we like this more". For instance in a recent project, the books with answers to questions had to be typeset in a multicolumn layout and because math was involved, we end up with lines with more height and depth than normal. That can not only result in more pages but also can make the result look a bit messy.

This paragraph demonstrates how lines are handled: when a paragraph is broken into lines each line becomes a horizontal box with a height and depth determined by the size of the characters that make up the line. There is a minimal distance between baselines (`baselineskip`) and when lines touch there can optionally be a `\lineskip`. In the end we get a vertical list of boxes and glue (either of not flexible) mixed with penalties that determine optimal paragraph breaks. This paragraph shows that there is normally enough space available to do the job.

We already have some ways to control this. For instance the dimensions of math can be limited a bit and lines can be made to snap on a grid (which is what publishers often want anyway). However, another alternative is to look at the line and decide if successive lines can be moved closer, of course within the constraints of the height and and depth of the lines. There is no real way to see if some ugly clash can happen simply because when we run into boxed material there can be anything inside and the dimensions can be set on purpose. This means that we have to honour all dimensions and only can mess around with dimensions when we're reasonably confident. In ConTEXt this messing is called profiling and that is what we will discuss next.

wider unprofiled

Regelmatig kom je procenten tegen. 'Pro centum' is Latijn en betekent per honderd, dus één van elke honderd, dus $\frac{1}{100}$ deel. Met procenten rekenen is daarom rekenen met honderdsten: 45% = $\frac{45}{100} = 0,45$. Dus 45% van een geheel is het $\frac{45}{100}$ deel ervan en dat kun je berekenen door te vermenigvuldigen met $0,45$.

shorter unprofiled

Regelmatig kom je procenten tegen. 'Pro centum' is Latijn en betekent per honderd, dus één van elke honderd, dus $\frac{1}{100}$ deel. Met procenten rekenen is daarom rekenen met honderdsten: 45% = $\frac{45}{100} = 0,45$. Dus 45% van een geheel is het $\frac{45}{100}$ deel ervan en dat kun je berekenen door te vermenigvuldigen met $0,45$.

example 1

wider unprofiled

Je gaat uit van de bekende eigenschappen van machten. Bijvoorbeeld: $g^r * g^s = g^{(r+s)}$. Neem je hierin $r = ^g \log(a)$ en $s = ^g \log b$, dan vind je: $g^{g \log(a)+g \log(b)} = g^{g \log a} \times g^{g \log b} = a \times b$. Hierbij gebruik je de definitieformules.

shorter unprofiled

Je gaat uit van de bekende eigenschappen van machten. Bijvoorbeeld: $g^r * g^s = g^{(r+s)}$. Neem je hierin $r = ^g \log(a)$ en $s = ^g \log b$, dan vind je: $g^{g \log(a)+g \log(b)} = g^{g \log a} \times g^{g \log b} = a \times b$. Hierbij gebruik je de definitieformules.

example 2

wider unprofiled

Omdat volgens de eigenschappen van machten en exponenten geldt $\frac{1}{x^4} = x^{-4}$ is ook hier sprake van een machtsfunctie, namelijk $f(x) = \frac{6}{x^4} = 6 \times \frac{1}{x^4} = 6x^{-4}$.

shorter unprofiled

Omdat volgens de eigenschappen van machten en exponenten geldt $\frac{1}{x^4} = x^{-4}$ is ook hier sprake van een machtsfunctie, namelijk $f(x) = \frac{6}{x^4} = 6 \times \frac{1}{x^4} = 6x^{-4}$.

example 3

Figure 7: Unprofiled examples.

3.1 Line heights and depths

In this section we will use some (Dutch) examples from documents that we've processed. We show unprofiled versions, with two different paragraph widths, in figure 7. All three examples shown demonstrate that as soon as we use something more complex than a number or variable in a subscript we exceed the normal line height, and thus the line spacing becomes somewhat irregular.

The profiled rendering of the same examples are shown in figure 8. Here we use the minimal heights and depths plus a minimum distance of 1pt. This default method is called `strict`.

In the first and last example there are some lines where the depth of one line combined with the height of the following exceeds the standard line height. This forces TEX to insert `\lineskip` (mentioned in the demonstration paragraph above),

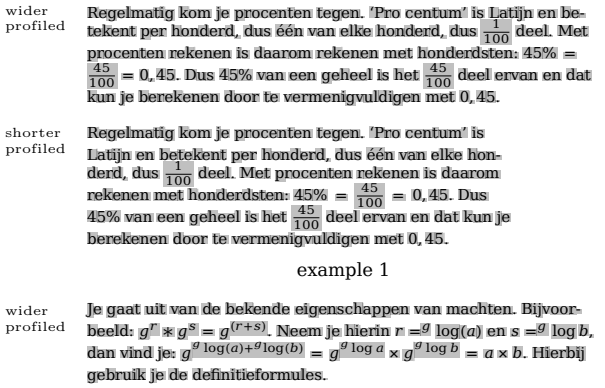


Figure 8: Profiled examples.

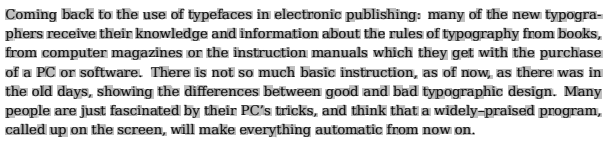

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

Figure 9: Normal lines profiled (quote from Hermann Zapf)

a dimension that is normally set to a fraction of the line spacing (for instance 1pt for a 10pt body font and 12pt line spacing). When we are profiling, `\lineskip` is ignored and we use a settable distance instead. The second example (with superscripts) normally comes out fine as the math stays within limits and we make sure that smaller fractions and scripts stay within the natural limits of the line, but nested scripts can be an issue.

In figure 9 we have profiled regular text, without math. Typical text stays well within the limits of height and depth. If this doesn't happen for prose then you need to adapt the height/depth ratio to the ascender/descender ratio of the bodyfont. Thus, for regular text it makes no sense to use the profiler, it only slows down typesetting.

3.2 When lines exceed boundaries

Let's now take a more detailed look at what happens when lines get too high or low. First we'll zoom in on a simple example: in figure 10, we compare a

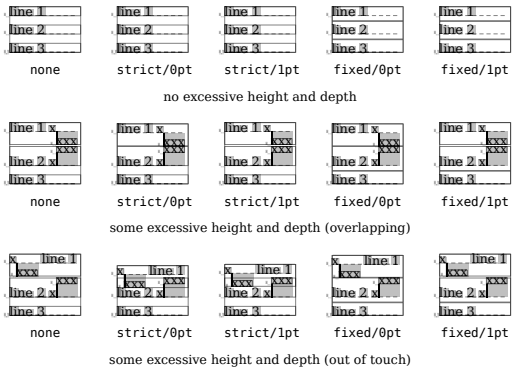


Figure 10: Variants of profiling, using a constructed two-line text.

sample text rendered using the variants of profiling currently implemented. (This is still experimental code so there might be more in the future). Seeing profiles helps to get a picture of the complications we have to deal with. In addition to the normal `vbox` variant (used in the previous examples), we show `none` which only analyzes, `strict` that uses the natural dimensions of lines and `fixed` that is supposed to cooperate with grid snapping.

Figure 10 shows what happens when we add some more excessive height and depth to lines. The samples are:

```
line 1 x\lower2ex\hbox{xxx}\par
line 2 x\raise2ex\hbox{xxx}\par
line 3 \par
```

and:

```
x\lower2ex\hbox{xxx} line 1 \par
line 2 x\raise2ex\hbox{xxx}\par
line 3 \par
```

Here the `strict` variant has some effect while `fixed` only has some influence on the height and depth of lines. Later we will see that `fixed` operates in steps and the default step is large so here we never meet the criteria for closing up.[4]

A profiled box is typeset with `\profiledbox`. There is some control possible but the options are not yet set in stone so we won't use them all here. Profiling can be turned on for the whole document with `\setprofile` but I'm sure that will seldom happen, and these examples show why: one cannot beforehand say if the result looks good. Let's now apply profiling to a real text. If you play with this yourself you can show profiles in gray with a tracker:

```
\enabletrackers[profiling.show]
```

We show the effects of setting distances in figure 11. We start with a zero distance:

[4] In ConTEXt we normally use `\high` and `\low` and both ensure that we don't exceed the natural height and depth.

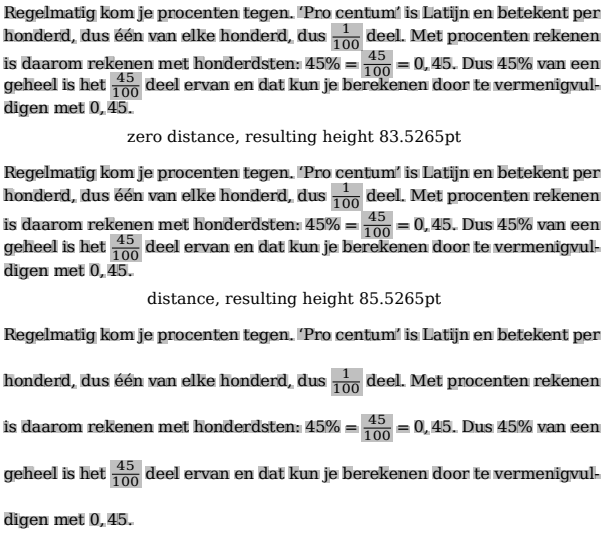


Figure 11: Example with different dimensions.

```
\profiledbox
  [strict]
  [distance=0pt]
  {\nl\getbuffer[example-1]}
```

Because we don't want lines to touch we then set the minimum distance to a reasonable value (1pt).

```
\profiledbox
  [strict]
  [distance=1pt]
  {\nl\getbuffer[example-1]}
```

Finally we also double the height and depth of lines, something that normally will not be done. The defaults are the standard height and depth (the ones you get when you inject a so-called `\strut`).

```
\profiledbox
  [strict]
  [height=2\strutht,
   depth=2\strutdp,
   distance=1pt]
  {\nl\getbuffer[example-1]}
```

The problem with this kind of analysis is that deciding when and how to use this information to improve spacing is non-trivial. One of the characteristics of user demand is that it nearly always concerns rather specific situations and that suggested solutions could work only in those cases. But as soon as we have one exceptional situation, intervention is needed which in turn means that a mechanism has to be under complete user control. That itself assumes that the user still has control, which is not the case in automated workflows. In fact, as soon as one is in control over the source and rendering, there are often easier ways to deal with the few cases that need treatment. Possible interference can come from, for instance:

- whitespace between paragraphs
- section titles (using different fonts and spacing)
- descriptions and other intermezzos
- images that interrupt the flow, or end up next to text
- ornaments like margin words (we catch some)
- text backgrounds making spacing assumptions

After a few decades of using TEX and writing solutions, it has become pretty clear that fully automated typesetting is a dream, if only because the input can be pretty weird and inconsistent and demands (from those who are accustomed to tweaking manually in a desktop publishing application) can be pretty weird and inconsistent too. So, the only real solution is to use some kind of artificial intelligence that one can feed with demands and constraints and that hopefully is clever enough to deal with the inconsistencies. As this kind of computing is unlikely to happen in my lifetime, poor man explicit solutions have to do the job for now. One can add all kinds of heuristics to the profiler but this can backfire when control is needed. Alternatively one can end up with many options like we have in grid snapping.

3.3 Where to use profiling

In ConTEXt there are four places (maybe a few more eventually) where this kind of control over spacing makes sense:

- the main text flow in single column mode
- multi-column mode, especially mixed columns
- framed texts, used for all kinds of content
- explicitly (balanced) split boxes

Because framed texts are used all over, for instance in tables, it means that if we provide control over spacing using profiles, many ConTEXt mechanisms can use it. However, enabling this for all packaging has a significant overhead so it has to be used with care so that there is no performance hit when it is not used. Here is an easy example using `\framed`:

```
\framed
  [align=normal,
   profile=fixed,
   frame=off]
  {some text ...}
```

For the following examples we define this helper:

```
\starttexdefinition demo-profile-1 #1
  \framed
    [align=normal,profile=#1]
    {xxx$\frac{1}{\frac{1}{\frac{1}{2}}}$
     \par
     $\frac{\frac{1}{\frac{1}{2}}}{2}$xxx}
\stoptexdefinition
```

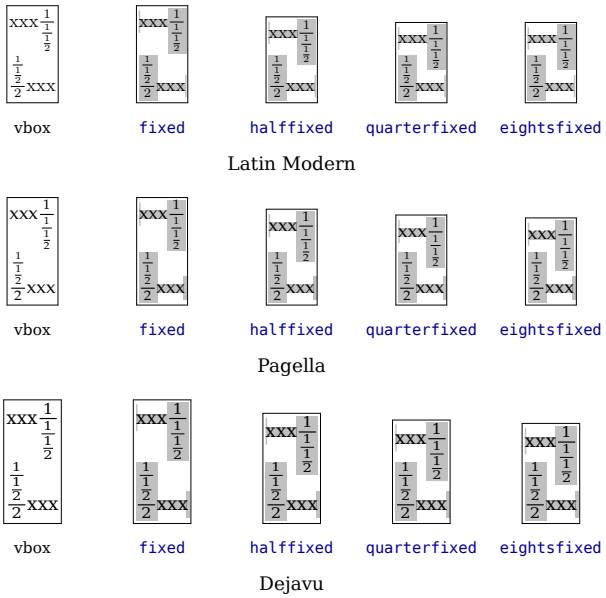


Figure 12: A few fonts compared.

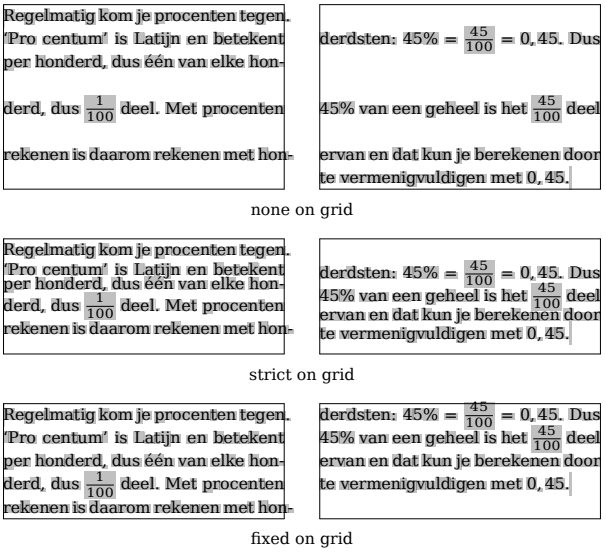


Figure 13: Boxed columns without and with profiling.

We apply this to predefined profiles. The macro is called like this:

```
\texdefinition{demo-profile-1}{fixed}
```

The outcome can depend on the font used: in figure 12 we show Latin Modern, TEX Gyre Pagella and Dejavu. Because in ConTEXt the line height depends on the bodyfont; each case is different.

As mentioned, we need this kind of profiling in multi-column typesetting, so let us have a look at that now. Columns are processed in grid mode but this is taken into account. We can simulate this by using boxed columns; see figure 13. One of the biggest problems is what to do with the bottom and top of a page or column. This will probably take a bit more to get right, and likely we will end up with different strategies. We can also think of special handlers but that will come with a high speed penalty. In the `strict` variant we don't mess with the dimension of a line too much, but the `fixed` alternative will get some more control.

Although using this feature looks promising it is also dangerous. For instance a side effect can be that interline spacing becomes inconsistent and even ugly. It really depends on the content. Also, as soon as some grid snapping is used, the gain becomes less, simply because the solution space is smaller. Then of course there is the matter of overall look and feel: most documents that need this kind of magic look bad anyway, so why bother. In this respect it is comparable to applying protrusion and expansion. There are hardly any combinations of design and content where micro-typography makes sense to use: in prose perhaps, but not in mixed content. On the other hand, profiling makes more sense in mixed content than in prose.

Not everything that is possible should be used. In figure 14 we show some fake paragraphs with profiles applied, the first series (random range 2) has a few excessive snippets, the last one (random range 5) has many. In figure 15 we show them in a different arrangement. Although there are differences it is hard to say if the results look better. We scaled down the results and used gray fake blurs instead of real text in order to get a better impression of the so-called (overall) grayness of a text.

3.4 Conclusion

Although profiling seems interesting, in practice it does not have much value in an automated flow. Ultimately, in the project for which I investigated this trickery, only in the final stage was some last minute optimization of the rendering done. We did that by injecting directives. Think of page breaks that make the result look more balanced. Optimizing image placement happens in an earlier stage because the text can refer to images like "in the picture on the left, we see ...". Controlling profiles is much harder. In fact, the more clever we are, the harder it gets to beat it when we want an exception. All these mechanisms: spacing, snapping, profiling, breaking pages, image placement, to mention a few, have to work together. For projects that depend on such placement, it might be better to write dedicated mechanisms than to try to fight with clever built-in features.

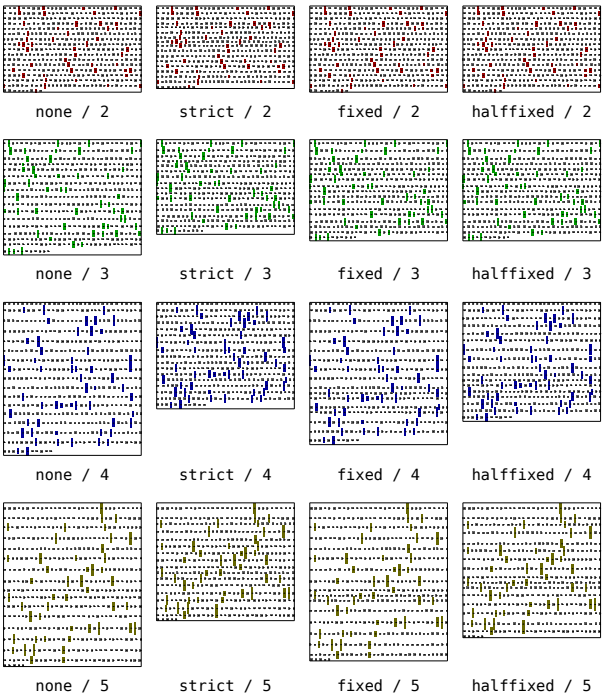


Figure 14: Gray examples; each row has progressively more excessive snippets.

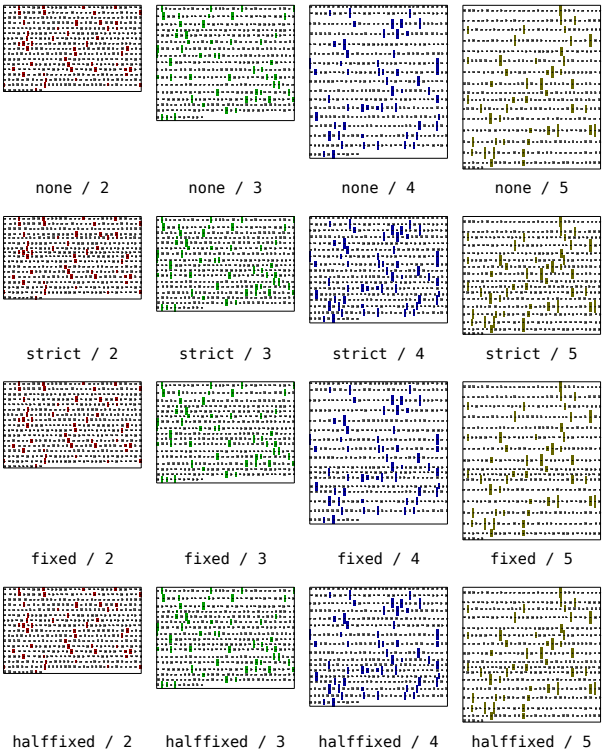


Figure 15: The same examples, rearranged such that each row has a different profiling variant.

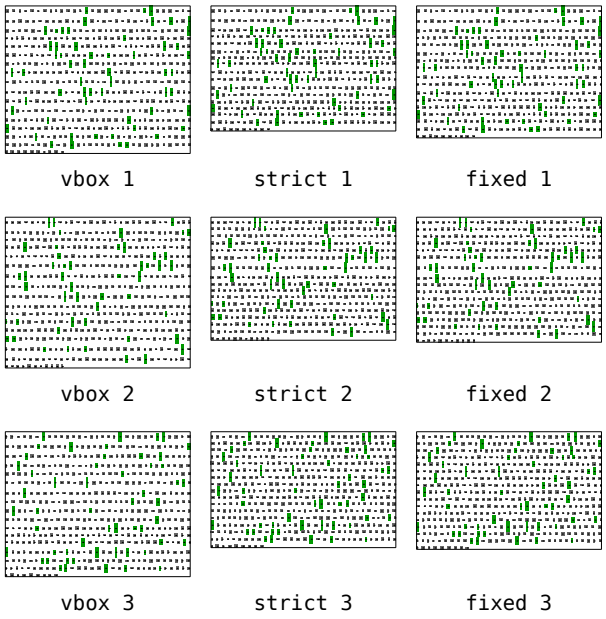


Figure 16: Three similar random cases.

In practice, probably only the `fixed` alternative makes sense and as that one has a boundary condition similar to (or equal, depending on other settings) snapping on gridsteps, the end result might not be that different from doing nothing. In figure 16 you see that the vbox variant is not that bad. And extremely difficult content is unlikely to ever look perfect unless some manual intervention happens. Therefore, from the perspective of "fine points of text typesetting" some local (manual) control might be more interesting and relevant.

In the end, I didn't need this profiling feature at all: because there are expectations with respect to how many pages a book should have, typesetting in columns was not needed. It didn't save that many pages, and the result would never look that much better, simply because of the type of content. Large images were also spoiling the game. Nevertheless we will keep profiles in the core and it might even get extended. One question remains: at what point do we stop adding such features? The answer would be easier if TEX wasn't so flexible.

◇ Hans Hagen
Pragma ADE
`http://pragma-ade.com`

TUG 2015 abstracts

Editor's note: Slides and other related information for many of the talks are posted at `http://tug.org/tug2015/program.html`.

– – ∗ – –

Kaveh Bazargan and Jagath AR
TEX — After 35 years, still the best solution for modern publishing
TEX is around 35 years old. The engine has remained almost unchanged. Since about 1990, the desktop publishing revolution gradually took the focus away from tags or mark-up in text, but the ubiquitous requirement for XML by publishers has focused attention on mark-up yet again, and TEX has returned as the king of automated pagination. We will discuss and demonstrate the advantages of TEX for pagination of XML over other pagination systems, including: fully automated pagination of XML files; highest typographic quality; production of "enhanced" PDFs, not possible by other means; obtaining different PDF styles from a single source.

Julien Cretel
Functional data structures in TEX
Because TEX lacks rich data structures, implementing even simple yet useful algorithms in it can be laborious. However, TEX is, in many ways, remarkably similar to functional programming languages, which are often praised for their expressive power.

Building on Alan Jeffrey's approach to embedding elements of the lambda calculus in TEX (see Alan's `lazylist` package), I plan to demonstrate how to implement and use richer data structures (such as binary search trees) in TEX & friends.

Olaf Drümmer
PDF/UA — what it is, how users can benefit from it, and how to get it right
PDF/UA is the latest addition to the group of international PDF standards. Published in 2012, it defines what a tagged PDF — as defined in PDF 1.7 (per ISO 32000-1) — must look like to be considered 'universally accessible', and how PDF/UA conforming tools should take advantage of its features. "Accessible" is often thought of as content accessibility from the point of view of people with some disability, but is not nominally limited to that. Content in a PDF/UA conforming file can also be more easily and more meaningfully accessed by software, allowing for intelligent content extraction or flexible repurposing (think formatted text reflow on mobile devices).

This talk gives a very compact overview of the rules defined in the PDF/UA standard, and how a PDF/UA file typically differs from an ordinary PDF file. Several sample usage scenarios will be demonstrated so attendees can get a feeling for how PDF/UA matters to users who have to rely on PDF/UA conforming documents and on suitable tools. Finally, several challenges will be discussed that document authors tend to run into.

Paul Gessler
Pretty-printing Git commit history graphs with PGF/TikZ
Increasing popularity of the distributed version control system Git has created a desire to integrate its versioning metadata into documents dynamically. An existing package, `gitinfo2`, by Brent Longborough, provides hooks and tools to access this information within LaTeX documents. My experimental package `gittree` adds a convenient interface for producing commit history graphs within LaTeX documents using the PGF/TikZ graphics language. I will present several examples of `gittree`'s use and discuss continuing development efforts.

Hans Hagen
What if . . .
What TEX does and what TEX doesn't do, the way a macro package is set up, how users use a TEX-like system, and what they expect (or demand) to a large extent depends on the circumstances in which the system was developed. In that respect a macro package is a snapshot of how at a certain moment texts ended up on the media popular at that time. How would TEX stand in a future museum? What if the developments around computer technology, the ideas about communication, the expectations of users and commerce had been slightly different? What if . . .

Bogusław Jackowski, Piotr Strzelczyk, and Piotr Pianowski
All the characters we need
We will discuss the choice of characters for math fonts.

Bogusław Jackowski, Piotr Strzelczyk, and Piotr Pianowski
Six GUST e-foundry math fonts and what next?
Since the publication of the math extension of the OpenType font format in 2007, barely a dozen OpenType math fonts have been released. This probably means that new math fonts are not (urgently) needed, which does not mean that existing fonts need not be improved, nor that creating special varieties of math fonts, such as heavy or sans serif variants, is useless.

Die TEXnische Komödie 2–3/2015

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`http://www.dante.de`). (Non-technical items are omitted.)

Die TEXnische Komödie 2/2015

Die TEXnische Komödie 3/2015

or tables when the inclusion is restricted to those chapters that actually have a float environment.

Rainer-M. Fritsch, Mehr Flexibilität mit den Variablen der Briefklasse `scrlttr2` [More flexibility with the variables of `scrlttr2`]; pp. 60–64

Extended discussion on `scrlttr2` variables.

Rolf Niepraschk, Briefkopien mit `scrlttr2` leicht gemacht [Creating letter copies with `scrlttr2`]; pp. 65–67

Often one wants to have a copy of a letter set with LaTeX. The simplest approach is to reprint the document, while it takes more effort to add a "copy" remark. This article shows how such "copy" versions can be created automatically.

[Received from Herbert Voß.]

***Eutypon* 32–33, October 2014**

Eutypon is the journal of the Greek TeX Friends (`http://www.eutypon.gr`).

Yiannis Mamaïs, 50 years of the Greek publishing house 'Gutenberg'; pp. 1–5

The Athenian publishing house "Gutenberg" celebrated its 50th anniversary in 2014. On that occasion, the author — once typographer and now book editor — gave a talk about issues of Greek book æsthetics at the bookstore Ianos in Athens, on 20 January 2014. The present article is that talk. (*Article in Greek with English abstract.*)

Apostolos Syropoulos, Pandoc: A Swiss Army knife for file conversion; pp. 7–12

The ability to convert markup files from one format to another is very important. Pandoc is an application written in the functional programming language Haskell which makes the conversion of markup files easy. Moreover, it allows for the management of bibliographic data. (*Article in Greek with English abstract.*)

Apostolos Syropoulos, Spot colors with (XeLaTeX; pp. 13–18

The term *spot color* refers to colors that are printed with their own ink. Typically, printshops use spot colors in the production of books or other printed material. The xespotcolor package allows the use of spot colors in documents prepared with XeLaTeX or LaTeX provided that the `dvipdfmx` driver is used. This article is a presentation of the capabilities of that package. It is worth noting that until recently, the capability to use spot colors was available only to those who prepare their documents with pdfLaTeX. (*Article in Greek with English abstract.*)

Ioannis A. Vamvakas, Greek crossword puzzles with LaTeX; pp. 19–24

This article presents a variation of the cwpuzzle package for the creation of Greek crossword puzzles. (*Article in Greek with English abstract.*)

Ioannis Dimakos and Dimitrios Filippou, Twenty-five years of Greek TeXing; pp. 25–34

The authors present an updated view of all available tools (fonts, systems and more) for typesetting Greek texts with (LA)TeX. Unlike the early days of Greek (LA)TeX, when the available tools were limited, users now have an abundance of tools. In addition, the emergence of Unicode-aware systems such as XeTeX and LuaTeX has allowed for a major breakthrough in the world of TeX: the use of OpenType and system fonts (i.e., fonts used by the operating system of the computer) for typesetting Greek texts. (*Article in English.*)

Dimitrios Filippou, TeXniques: Games with fonts; pp. 35–38

This regular column shows when and how to use Linux Libertine numerals in XeLaTeX math mode, and how to get true Greek small caps from the same font family. (*Article in Greek.*)

Dimitrios Filippou, Book presentations; pp. 39–42

The following books — all in Greek — are presented:
(a) Martin Davies, *Aldus Manutius: Printer and Publisher of Renaissance Venice*, Greek translation by Toula Sioti, Libro Editions, Athens 2004;
(b) Ioannis K. Mazarakis-Ainian, *The Greek Printshops during the Greek Independence War of 1821–1827*, National Historical Museum, Athens 2007;
(c) Yannis Kokkonas (editor), *Typographers and Typography*, Proceedings of a Memorial Symposium for the Typographer Christos G. Manousaridis, National Hellenic Research Foundation, Athens 2013; and
(d) Panayiotis Haratzopoulos, *Ten Plus One Font Designers, Ten Plus One Historic Fonts*, Gramma Editions, Athens 2013.
(*Article in Greek.*)

[Received from Dimitrios Filippou and Apostolos Syropoulos.]

TUG 2015 election report

Kaja Christiansen
for the Elections Committee

As has been previously announced electronically, the election results have been tallied. Kaveh Bazargan has been elected TUG president for the term that ends with the 2017 annual meeting. The following votes were counted:

Kaveh Bazargan, 307
Jim Hefferon, 110

Both candidates made a good showing, although the total number of voters was only about a third of those eligible.

As of the ballot closing date (May 11), 145 valid paper votes and 291 valid electronic ballots were received. Of the 291 electronic votes, 18 were repeats of the same vote. One member voted (differently) by paper after voting electronically, so, per our election procedures, the electronic vote was excluded.

A script was written to assist in processing the electronic votes. Mailing the paper votes and matching these up with electronic votes was time-consuming and expensive. We suggest that the electronic voting may be strongly encouraged in the future, while paper ballots could be sent by request. We will make specific recommendations for changes to the election procedures in due course.

As previously announced, the number of candidates for open board positions was fewer than positions, so no ballot was required. Those results, and all candidates' statements, were printed in the *TUGboat* 36:1.

The Committee gratefully acknowledges the diligent work of the TUG executive director, Robin Laakso, in administering all aspects of the election, from inviting nominations to the final tally.

This was the first contested election since 2005. Thanks to everyone for their participation.

◇ Kaja Christiansen
for the Elections Committee
`http://tug.org/election`

TUG Institutional Members

American Mathematical Society, *Providence*, *Rhode Island*

Aware Software, Inc., *Midland Park*, *New Jersey*

Center for Computing Sciences, *Bowie*, *Maryland*

CSTUG, *Praha*, *Czech Republic*

Fermilab, *Batavia*, *Illinois*

Google, *San Francisco*, *California*

IBM Corporation, T J Watson Research Center, *Yorktown*, *New York*

Institute for Defense Analyses, Center for Communications Research, *Princeton*, *New Jersey*

Masaryk University, Faculty of Informatics, *Brno*, *Czech Republic*

MOSEK ApS, *Copenhagen*, *Denmark*

New York University, Academic Computing Facility, *New York*, *New York*

River Valley Technologies, *Trivandrum*, *India*

River Valley Technologies, *London*, *United Kingdom*

RSGP Consulting Pvt. Ltd., *Trivandrum*, *India*

ShareLaTeX, *United Kingdom*

Springer-Verlag Heidelberg, *Heidelberg*, *Germany*

StackExchange, *New York City*, *New York*

Stanford University, Computer Science Department, *Stanford*, *California*

Stockholm University, Department of Mathematics, *Stockholm*, *Sweden*

TNQ, *Chennai*, *India*

University College, Cork, Computer Centre, *Cork*, *Ireland*

Université Laval, *Ste-Foy*, *Québec*, *Canada*

University of Cambridge, Centre for Mathematical Sciences, *Cambridge*, *United Kingdom*

University of Ontario, Institute of Technology, *Oshawa*, *Ontario*, *Canada*

University of Oslo, Institute of Informatics, *Blindern*, *Oslo*, *Norway*

University of Wisconsin, Biostatistics & Medical Informatics, *Madison*, *Wisconsin*

VTEX UAB, *Vilnius*, *Lithuania*

TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `http://tug.org/consultants.html`. If you'd like to be listed, please see that web page.

Aicart Martinez, Mercè
Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) ono.com`
Web: `http://www.edilatex.com`
We provide, at reasonable low cost, LaTeX or TeX page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

Dangerous Curve
PO Box 532281
Los Angeles, CA 90053
+1 213-617-8483
Email: `typesetting (at) dangerouscurve.org`
Web: `http://dangerouscurve.org/tex.html`
We are your macro specialists for TeX or LaTeX fine typography specs beyond those of the average LaTeX macro package. If you use XeTeX, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical TeX and LaTeX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a TeX book.

Latchman, David
4113 Planz Road Apt. C
Bakersfield, CA 93309-5935
+1 518-951-8786
Email: `david.latchman (at) texnical-designs.com`
Web: `http://www.texnical-designs.com`
LaTeX consultant specializing in: the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized packages to meet your needs.

Call or email to discuss your project or visit my website for further details.

Peter, Steve
+1 732 306-6309
Email: `speter (at) mac.com`
Specializing in foreign language, multilingual, linguistic, and technical typesetting using most flavors of TeX, I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. In addition, I've helped publishers write, maintain, and streamline TeX-based publishing systems. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Sievers, Martin
Im Alten Garten 5
54296 Trier, Germany
+49 651 4936567-0
Email: `info (at) schoenerpublizieren.com`
Web: `http://www.schoenerpublizieren.com`
As a mathematician with ten years of typesetting experience I offer TeX and LaTeX services and consulting for the whole academic sector (individuals, universities, publishers) and everybody looking for a high-quality output of his documents. From setting up entire book projects to last-minute help, from creating individual templates, packages and citation styles (BibTeX, biblatex) to typesetting your math, tables or graphics — just contact me with information on your project.

Sofka, Michael
8 Providence St.
Albany, NY 12203
+1 518 331-3457
Email: `michael.sofka (at) gmail.com`
Skilled, personalized TeX and LaTeX consulting and programming services.

I offer over 25 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in TeX and LaTeX: Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in TeX or LaTeX; Generating custom output in PDF, HTML and XML; Data format conversion; Databases.

If you have a specialized TeX or LaTeX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris
46871 Antioch Pl.
Sterling, VA 20164
+1 703 915-2406
Email: `borisv (at) lk.net`
Web: `http://www.borisv.lk.net`
TeX and LaTeX consulting, training and seminars. Integration with databases, automated document preparation, custom LaTeX packages, conversions and much more. I have about eighteen years of experience in TeX and three decades of experience in teaching & training. I have authored several packages on CTAN, published papers in TeX related journals, and conducted several workshops on TeX and related subjects.

Young, Lee A.
127 Kingfisher Lane
Mills River, NC 28759
+1 828 435-0525
Email: `leeayoung (at) morrisbb.net`
Web: `http://www.thesiseditor.net`
Copyediting your `.tex` manuscript for readability and mathematical style by a Harvard Ph.D. Your `.tex` file won't compile? Send it to me for repair. Experience: edited hundreds of ESL journal articles, economics and physics textbooks, scholarly monographs, LaTeX manuscripts for the Physical Review; career as professional, published physicist.

Calendar

2015

Sep 28 — *TUGboat* **36**:3, submission deadline.

Oct 14 – 17 — Association Typographique Internationale (ATypI) annual conference, Theme: "Challenges", São Paulo, Brazil. www.atypi.org

Oct 17 — GuIT Meeting 2015, XII Annual Conference, Trento, Italy. www.guitex.org/home/en/meeting

Oct 19 – 20 — The Thirteenth International Conference on Books, Publishing, and Libraries, "The Event of the Book", University of British Columbia, Vancouver, Canada. booksandpublishing.com/the-conference-2015

Oct 31 — UK-TUG Annual Meeting, Oxford, UK. uk.tug.org

Nov 6 – 10 — ASIS&T 2015 Annual Meeting, "Information Science with Impact: Research in and for the Community", American Society for Information Science and Technology, St. Louis, Missouri. www.asis.org/events/annual-meeting

---

2016

Feb 25 – 27 — Typography Day 2016, "Typography and Education", Srishti School of Art, Design & Technology, Bangalore, India. www.typoday.in

Jul 5 – 9 — The 6th International Conference on Typography and Visual Communication (ICTVC), "Against lethe ...", Thessaloniki. Greece www.ictvc.org

Jul 12 – 16 — Digital Humanities 2016, Alliance of Digital Humanities Organizations, "Digital Identities: the Past and the Future", Kraków, Poland. dh2016.org

---

TUG 2016
Toronto, Canada.

Jul 23, 24 — Optional pre-conference tours.

Jul 24 — Evening reception and registration.

Jul 25 – 27 — The 37th annual meeting of the TEX Users Group. Presentations covering the TEX world. tug.org/tug2016

Jul 28 — Optional typographic excursions and banquet.

Jul 29 — Optional post-conference tour [potential].

Jul 24 – 28 — SIGGRAPH 2015, "Render the Possibilities", Anaheim, California. s2016.siggraph.org

Sep 13 – 16 — ACM Symposium on Document Engineering, Vienna, Austria. www.doceng2016.org

Oct 14 – 18 — ASIS&T 2016 Annual Meeting, American Society for Information Science and Technology, Copenhagen, Denmark. www.asis.org/events/annual-meeting

Status as of 15 September 2015

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568. e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

User group meeting announcements are posted to lists.tug.org/tex-meetings. Interested users can subscribe and/or post to the list, and are encouraged to do so.

Other calendars of typographic interest are linked from tug.org/calendar.html.

Introductory

Intermediate

Intermediate Plus

Advanced

Contents of other TEX journals

Reports and notices