TUGboat

Volume 36, Number 1 / 2015

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: April 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`

You have patience for detail and iterative processes.
Kerning is a great time for meditation, really.

Tobias Frere-Jones
http://www.frerejones.com/about/jobs/

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 36, NUMBER 1 • 2015
PORTLAND • OREGON • U.S.A.

TUGboat editorial information

This regular issue (Vol. 36, No. 1) is the first issue of the 2015 volume year.

TUGboat is distributed as a benefit of membership to all current TUG members. It is also available to non-members in printed form through the TUG store (`http://tug.org/store`), and online at the *TUGboat* web site, `http://tug.org/TUGboat`. Online publication to non-members is delayed up to one year after print publication, to give members the benefit of early access.

Submissions to *TUGboat* are reviewed by volunteers and checked by the Editor before publication. However, the authors are still assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

Submitting items for publication

Proposals and requests for *TUGboat* articles are gratefully accepted. Please submit contributions by electronic mail to `TUGboat@tug.org`.

The second 2015 issue will be the proceedings of TUG'15 (`http://tug.org/tug2015`); the deadline for receipt of final papers is July 31. The third issue deadline is September 25.

The *TUGboat* style files, for use with `plain` TEX and LATEX, are available from CTAN and the *TUGboat* web site. We also accept submissions using ConTEXt. Deadlines, tips for authors, and other information: `http://tug.org/TUGboat/location.html`

Effective with the 2005 volume year, submission of a new manuscript implies permission to publish the article, if accepted, on the *TUGboat* web site, as well as in print. Thus, the physical address you provide in the manuscript will also be available online. If you have any reservations about posting online, please notify the editors at the time of submission and we will be happy to make special arrangements.

Ab Epistulis

Steve Peter

2105 is an election year, with seats for the Board and for President open. For the Board of Directors, the following individuals were nominated: Pavneet Arora, Barbara Beeton, Karl Berry, Susan DeMeritt, Michael Doob, Cheryl Ponchin, Norbert Preining, and Boris Veytsman. As there were not more Board nominations than open positions, all these nominees are duly elected to a four-year term. Thanks to all for their willingness to serve.

I'm pleased that two dedicated individuals have stepped forward to run for TUG President. Kaveh Bazargan is a long-time TEX enthusiast who has served as cinematographer at many a TUG meeting, ensuring that people who could not attend in person have access to the information presented there. Jim Hefferon is the current TUG Vice President and has served the community in a variety of capacities, most notably as one of the hosts and organizers for CTAN. Since there is but a single office of President, an election ballot is required, in accordance with the TUG election procedures. Details are online at `http://tug.org/election`; perhaps the main point is that this year, it is possible to vote electronically in the TUG members area, `https://www.tug.org/members`, until May 11.

Terms for President and members of the Board of Directors will begin with the Annual Meeting. Congratulations to all.

Taco Hoekwater, Ross Moore, and Philip Taylor have decided to step down at the end of this term. I wish to thank them for their service, and for their continued participation until the Annual Meeting. I will also step down at the end of my term, but I do intend to open the Annual Meeting as President one last time.

The 2015 TUG Annual Meeting will be held in Darmstadt, Germany, July 20–22. If you'd like to make a presentation at the conference, please see `http://tug.org/tug2015/cfp.html`. Early submissions are greatly appreciated. Please write us at `tug2015@tug.org` for any questions, expressions of interest, etc.

Book your hotel and flight, and I hope to see you there! See `http://tug.org/tug2015` for information.

Since the election technically makes me a lame duck (don't worry, I've been called worse!), I've been thinking about my tenure and the challenges that lie ahead. I am pleased that TUG as an organization is running quite smoothly now, which is due in no small part to the outstanding team I had to work with: my executive committee, Jim Hefferon, Sue DeMeritt, and Karl Berry, and especially the day to day work of our Executive Director, Robin Laakso.

The main challenge that I am disappointed I was not able to tackle is the shrinking membership numbers for TUG. I'm not worried about TEX in a global context, since DANTE and NTG are strong, and enthusiastic development continues on LATEX3 and ConTEXt, but I would love to see a groundswell in TEX usage and TUG support in the US. Thanks to Boris Veytsman, we do have a special TUG membership campaign this year: members invite members. Please see `http://tug.org/membership`. If you have additional concrete ideas to help us grow TUG, please let us know.

It has been an honor serving you. Happy TEXing!

◇ Steve Peter
Princeton University Press
`http://tug.org/TUGboat/Pres`

Editorial comments

Barbara Beeton

Status of CTAN at Cambridge

As of mid-February, the Cambridge CTAN node developed problems, and has been downgraded; at least for the time being, it should not be used.

This is a good opportunity to remind anyone who is accessing CTAN to enter the network via the established mirroring system. To search for a package, go to `http://www.ctan.org/search` and on selecting an item, a suitable mirror will be chosen to deliver it. Also, the top-level host name `http://mirror.ctan.org` will automatically redirect to such a suitable mirror.

Before his retirement from the Cambridge Computer Lab last fall, Robin Fairbairns was the on-site gatekeeper (for decades) for the Cambridge CTAN node. With his departure, management of the node is less certain, hence the reduction in its status.

But let us take this opportunity to thank Robin for his long years of devoted service to CTAN and to the UK TEX FAQ, and wish him a long, healthy and fulfilling retirement. Robin intends to continue with the FAQ, for which he has reported a sizeable backlog. It's in excellent hands.

RIP Brian Housley

Brian Housley's widow, Zora, has informed the TUG office that Brian passed away on 26 January 2015, and she requested that the news be communicated to the TEX community.

Brian was a resident of Switzerland, an emeritus professor from the Universität Bern. He had been a member of TUG since 1988, and attended several annual TUG conferences, accompanied by his wife.

Brian was the author of the `hletter` package, on CTAN and in TEX Live. He also wrote an article about `hletter` in *TUGboat* **32**:3, 302–308 (`http://tug.org/TUGboat/tb32-3/tb102housley.pdf`), based on his presentation at TUG 2011 in India.

We extend our condolences to Brian's widow.

Oh, zero! — Lucida news

Chuck Bigelow's article "Oh, oh, zero!" (*TUGboat* **34**:2 (2013), 168–181) elicited a reaction from Don Knuth (*TUGboat* **35**:3 (2014), 232–234), which has in turn elicited a further reaction from Chuck, in the form of reshaped capital 'O' and 'Q'. The reshaping takes a super-elliptical form, reminiscent of Piet Hein's superegg. Chuck has installed these as variant letters in Lucida Grande Mono, which is equivalent in design and proportions to Lucida Sans Typewriter. Don's actual request was to place the squarish shapes in Lucida Console; this was more involved, as it also required shortening the capitals to the Console height, but has now also been accomplished. This special "DEK edition" will also have a zero-slash as an alternate, the better for clarity in reading code in a typewriter font.

The Lucida Grande Mono and Lucida Console fonts with the new glyphs will eventually be made available only from TUG. (The non-enhanced versions will continue to be offered, only in TrueType format, from the Lucida Store, `http://lucidafonts.com`.) Stay tuned.

In a related development, Chuck and his partner, Kris Holmes, have posted on their blog (`http://bigelowandholmes.typepad.com`) a detailed essay, "How and Why We Designed Lucida", celebrating the 30th anniversary of these types, "the first family of original, digital typefaces for laser printing and screen displays". The salient characteristics of this family are presented in detail, along with a clear explanation of how these choices make Lucida different from other types, especially those meant originally for printing directly on paper. Many choices were guided by the results of research into the human visual system, and the rich background in calligraphy that Chuck and Kris share is also a strong influence.

Their blog in general is a veritable history lesson in all aspects of typography and vision research, and if it were less readily available, we would have coopted several items from there for publication in these pages. Expected soon is an article about "Women's Literation" (yes, that's "t", not "b"); be prepared for a surprise conclusion.

Another occasional feature of the `lucidafonts` site is the "Flash Sale", which appears irregularly for specific font families, and may similarly disappear without warning. The sales *are* announced on Twitter (`https://twitter.com/LucidaFonts`), and anyone who downloads a free font from the site gets an option to be put on a list for email announcements. A recent offering was Lucida Handwriting: 15 weights, $15. There's no telling what might be featured next.

And if that is not enough, `lucidafonts.com` contains many more features that are not yet easily accessible (probably, Chuck says, because the site navigation is not yet fully developed). For instance, on the `fonts/` page is a matrix of the letter "a" for every font in the store, showing weight and style variations. And on the `pages/facts` and `pages/faq` pages of the site are more interesting bits of technical and historical information. Explore!

First Annual Updike Prize

Last spring, the Daniel Berkeley Updike Collection at the Providence Public Library (PPL) was host to the inauguration of a new prize for student type designers (*TUGboat* **35**:1, page 3), with a talk by Matthew Carter. The first of the annual prizes were awarded this year on February 19; the ceremony also featured a talk by Tobias Frere-Jones.

From the Updike Prize web page (`http://www.provlib.org/updike-prize-finalists`):

> The annual Updike Prize rewards undergraduate and graduate type designers whose work has been influenced by materials in the Updike Collection at the Providence Public Library. Whether students choose to revive a historic typeface or to create a new typeface inspired by an earlier design, applications are judged on the quality of the specimen, the quality of the typeface submitted and how creatively and thoughtfully it interprets a historical model.

The finalists and their typefaces/type families:

- Sandra Carrera, *Pícara* (*ECAL*) First Prize
- Chae Hun Kim, *Hodoo*
- Prin Limphongpand, *Rizvele* (Runner-Up)
- Yeon Hak Ryoo, *Tranche*

Carrera's family, *Picara*, was influenced by a type specimen published in the 1770s by Antonio Espinosa. The entire book can be viewed on the PPL web site, at `https://pplspc.org/espinosa`.

The name of Prin's typeface, *Rizvele*, is an anagram of the source used, a book from the venerable Elzevir publishing house.

The first prize trophy, a fully functional composing stick, is pictured on the PPL's "Notes for Bibliophiles" blog, `https://pplspcoll.wordpress.com`, as part of the offering for February 26, 2015. First Prize also includes $250 and admission to the TypeCon 2015 conference (compliments of the Society of Typographic Aficionados).

Information about next year's Updike Prize competition will appear on the PPL blog in due time, but the rules are not likely to change significantly. It's not too early to get started.

Talk by Tobias Frere-Jones

The talk, on the occasion of the awarding of the Updike Prize, was titled "How I Got Here". It was also an occasion for Frere-Jones to revisit the scene of his formal typographic education, the Rhode Island School of Design (RISD).

Frere-Jones came by his interest in print naturally. His great-grandfather, Edgar Wallace, was a writer of books, including mysteries. (The slides included images of their jackets, from the time when books routinely wore jackets.) His father was a copywriter for an advertising agency.

At an early age, Frere-Jones became interested in art. He named two principal influences — Kazimir Malevich, a creator of geometric abstract art, and Kurt Schwitters, who made collages from found scraps of printed material. Another influence was mathematics, in particular algebra and geometry. Analyzing shapes — squares, circles and triangles — was the only way he could make sense of Roman inscriptional letters. He also found inspiration (and examples for practice) in a book on drawing letterforms by David Gates. After entering RISD, he augmented what was being covered in classes with Updike's classic *Printing Types*.

One early experiment, intended as a Christmas present (to be used in ads for his brother's band), involved drawing his preliminary sketches on napkins while at a local pub. The first draft included only letters, and when he realized that he also needed digits, he had to go back to the bar to get more napkins to establish the proper conditions for matching the shapes.

After graduation from RISD, he joined the Font Bureau, where he benefited from the experience of David Berlow, whose good lessons included how to say when a project is finished, and assignments where he could be effective.

Among the many typefaces created by Frere-Jones are Interstate and Retina (both well known, even ubiquitous). He also adapted for digital use a lesser-known face named Epitaph, originally created at the close of the nineteenth century and based on letterforms found on contemporary gravestones; it was this typeface, he said, that caused him to realize that lettershapes (in metal) were different in different sizes. He described his adaptations as taking parts of letters from different sizes, swapping features from one letter to another, for example, adding serifs on "T", transplanting a curve from "N" to "K". In the process, he produced two variants of almost every letter, based on features already present in the different sizes of the original. Although his slides showing the details aren't available, a good example is posted at `http://www.fontbureau.com/fonts/Epitaph`. Look for the subtle variations in the lettershapes of the two alternate alphabets. In this context, Frere-Jones also noted that a client may take him in a direction that he'd never have gone on his own.

The talk was followed by a few questions from the audience. Asked what is his favorite letter, he answered "R": it exhibits features from all the shapes he identified earlier — square, circle, and triangle. What annoys him most? The degree of impatience

shown by current students; things are too easy now, compared to cutting letters in metal — that takes time, and inspires reflection. He noted that Matthew Carter said that he never regretted putting something in a drawer and ignoring it for a year. And is he still having fun? Yes! Especially when he is able to just sit and draw.

Much more can be discovered on Frere-Jones' blog at `http://www.frerejones.com`.

Monotype Recorder online

For about 70 years, from 1902, the Monotype Corporation published a trade magazine, *The Monotype Recorder*. It covered topics of great variety concerning font specimens, "best practices" and typographic conventions for various purposes and in various locales, reviews of well known printing establishments (who were Monotype customers), a wealth of material about a skilled trade.

A large (and growing) collection of issues of the *Recorder* have now been scanned and posted as PDF files, available for download, on the Metal Type web site in the UK, at `http://www.metaltype.co.uk/monotype_recorder.shtml`.

One particular issue (Vol. 40, No. 4, Winter 1956) is devoted to the topic "Setting Mathematics", by Arthur Phillips, the acknowledged expert on the subject. This was apparently unknown to Knuth when he designed TEX, as it does not appear in any of his early bibliographies. But it sets forth the principles by which math compositors produced much of the material that Knuth examined in his design exploration.

Truly a trove worth treasuring!

Doves Press type recovered

This news has been reported in several places now, but it's no less amazing on a repeat reading: "A century after being cast into the River Thames, a celebrated typeface reemerges" (`http://hyperallergic.com/181625`).

Thomas Cobden-Sanderson and Emery Walker together produced stunning works of the printer's art at their Doves Press, using a type created for its exclusive use. After a falling out between the partners, the press closed in 1909, but its effects remained jointly owned. But before his death in 1917, Cobden-Sanderson, who could not bear the thought that his precious type might be used for anything less worthy than the work for which it had been created, spirited the entire collection to the edge of the Thames, and threw it in.

In 2010, type designer Robert Green decided to undertake a digital revival of the Doves Press type. Working from original examples and archival material, he produced a version that was released in 2013. The original type was still missing, but there were enough clues to locate its watery resting place under the Hammersmith Bridge. Searching on the bank at low tide, Green found several pieces of type in the mud, and from there, with the assistance of a salvage team from the Port of London Authority, 150 pieces of type were recovered.

That small sample of type is not enough to print a book, but it was enough to provide a few new details about the design, and validate the accuracy of the digital revival. An updated version of the facsimile was released in December 2014, and a very readable story, with copious illustrations, appears on the cited Hyperallergic web page.

Another account of the adventure can be found online at `http://trov.es/1zwdjHM`, "The Gorgeous Typeface That Drove Men Mad and Sparked a 100-Year Mystery".

Textures resurfaces

With Barry Smith's death in 2012 (*TUGboat* **34**:2, pages 111, 112), BlueSky Research folded its tents and work on upgrading *Textures* to run natively under Mac OS X came to a halt. Now, a group of dedicated academic users has banded together to support the completion of a Cocoa-based implementation of *Textures*, left unfinished at Barry's death, and to this end has made the existing Carbon-based implementation available for evaluation at no cost.

Preliminary details of this project have been posted at `http://blueskytex.com`, with a further announcement planned for July.

LaTeX vs. Word in academic publications

A question on the TeX.stackexchange forum, "Does LaTeX really perform worse than Word?" (`http://tex.stackexchange.com/q/219576/`), led me to a study published in a recent issue of *PLoS One* that concludes "that even novice MS Word users perform better than expert LaTeX users in document creation."

The article, written by Markus Knauff and Jelica Nejasmic, entitled "An Efficiency Comparison of Document Preparation Systems Used in Academic Research and Development" (`http://journals.plos.org/plosone/article?id=10.1371/journal.pone.011506`), not surprisingly, raised a number of questions, not to mention a furore, in a number of web discussions. These are only a few:

- comments on the *PLoS* site (click on the "Reader comments" button on the article page);
- the TeX.sx chat: `http://chat.stackexchange.com/rooms/19762`;

- *Nature*: "Word-processing war flares up on social media" (http://www.nature.com/news/word-processing-war-flares-up-on-social-media-1.16669).

Admittedly, readers of *TUGboat* may be predisposed in favor of LaTeX. Nonetheless, my main gripe with the article is the "conclusion". While admitting that the use of LaTeX may be justified when documents contain a large amount of math, the final statement is this:

> In all other cases, we think that scholarly journals should request authors to submit their documents in Word or PDF format. [...] preventing researchers from producing documents in LaTeX would save time and money to maximize the benefit of research and development for both the research team and the public.

"Preventing" — so much for academic freedom. (This may have been peer reviewed, but my opinion of the quality of that review isn't favorable.)

Miscellanea

Here are a few more things I found interesting.

Most of the presentations at last year's BachoTeX have slides linked from the program: http://www.gust.org.pl/bachotex/2014/program. This one definitely lives up to its title, "Absolutely non-computer and completely not programmable new book forms", by Andrzej Tomaszewski (http://www.gust.org.pl/bachotex/2014-pl/presentations/formy-ksiazki.pdf). Fascinating illustrations!

Slides and recordings of the UK TUG 2013 and 2014 meetings are linked from their home page, http://uk.tug.org. A number of interesting topics are covered.

A look at the persistence of "archaic" typographic practices was featured in the Boston Globe: "Modern typefaces vs. the Massachusetts court system: Why does one of the nation's most progressive states do its legal business in old-fashioned Courier?" (http://tinyurl.com/p232hva). The image of an ancient typewriter heads the article. (This typewriter is very like the one I learned to type on in summer school. Even then it was an anachronism, from long before Courier was created, in 1955. Other anachronisms in the article are so prevalent that it should have been held for publication until April 1!) When this was mentioned to Chuck Bigelow, he responded with a reference to an article he wrote with Gordon Legge published in the *Journal of Vision* (http://www.journalofvision.org/content/11/5/8.long); monospace fonts are still used in Hollywood scripts:

> ... fixed-width Courier, which provides a convenient, predictable metric. Each standard script page corresponds to roughly 1 min of movie time, so the number of pages gives producers an estimate of the length and production cost of a movie.

"Overheard" on the TeX.stackexchange chat: "Archimedes principle: the upward force by the text on a float is equal to the weight of the text the float displaces. :)" (The perpetrator will not be identified here, to protect the guilty.)

Google Code, an open-source project hosting service started in 2005, has been home to some TeX-related projects. Effective 12 March 2015, the service has disabled new project creation, and 25 January 2016 has been announced as the closing date. The full announcement can be found at http://google-opensource.blogspot.co.uk/2015/03/farewell-to-google-code.html. At least one affected code developer has migrated his code to GitHub (https://github.com), but other sites are possible. If your code has been hosted at Google Code, you probably already know about this change; if you are looking for something that has been hosted there, and are unsure of its new location, it's probably best to check with the developer.

A final admonishment

There is much going on in the TeX world. I am tuned into a number of the (La)TeX forums and mailing lists, and the level of activity is high. Along with the usual plethora of beginners' questions, interesting topics of general interest abound, many worthy of more in-depth and polished coverage. If you are responsible for such material: please consider writing it up and submitting it for publication here.

If you are only an observer of interesting material, rather than the originator, and would like to see it addressed in print, send a note to the editors citing the source (preferably with reliable links), and we will endeavor to persuade the perpetrators to write it up for publication.

Another occasional feature that we've missed in *TUGboat* is cartoons and other illustrations on typographic subjects. If you see a drawing or photo that fits our "profile", let us know, and we'll try to persuade the creator to become a contributor.

TUGboat is your journal. Enjoy reading it, surely — but please also think about becoming an active participant.

◇ Barbara Beeton
http://tug.org/TUGboat
tugboat (at) tug dot org

Hyphenation exception log

Barbara Beeton

This is the periodic update of the list of words that TEX fails to hyphenate properly. The full list last appeared in *TUGboat* 16:1, starting on page 12, with updates in *TUGboat* 22:1/2, pp. 31–32; 23:3/4, pp. 247–248; 26:1, pp. 5–6; 29:2, p. 239; 31:3, p. 160; 33:1, pp. 5–6; and 34:2, pp.113-114.

In the list below, the first column gives results from plain TEX's `\showhyphens{...}`. The entries in the second column are suitable for inclusion in a `\hyphenation{...}` list.

In most instances, inflected forms are not shown for nouns and verbs; note that all forms must be specified in a `\hyphenation{...}` list if they occur in your document. The full list of exceptions, as a TEX-readable file, appears at `http://mirror.ctan.org/info/digests/tugboat/ushyphex.tex`. (It's created by Werner Lemberg's scripts, available in the subdirectory `hyphenex`.)

Like the full list, this update is in two parts: English words, and one non-English name that occurs in English (mathematical) texts.

Thanks to all who have submitted entries to the list. Here is a short reminder of the relevant idiosyncrasies of TEX's hyphenation. Hyphens will not be inserted before the number of letters specified by `\lefthyphenmin`, nor after the number of letters specified by `\righthyphenmin`. For U.S. English, `\lefthyphenmin=2` and `\righthyphenmin=3`; thus no word shorter than five letters will be hyphenated. (For the details, see *The TEXbook*, page 454.) This particular rule is violated in some of the words listed; however, if a word is hyphenated correctly by TEX except for "missing" hyphens at the beginning or end, it has not been included here.

Some other permissible hyphens have been omitted for reasons of style or clarity. While this is at least partly a matter of personal taste, an author should think of the reader when deciding whether or not to permit just one more break-point in some obscure or confusing word. There really are times when a bit of rewriting is preferable.

One other warning: Some words can be more than one part of speech, depending on context, and have different hyphenations; for example, 'analyses' can be either a verb or a plural noun. If such a word appears in this list, hyphens are shown only for the portions of the word that would be hyphenated in the same way regardless of usage.

The reference used to check these hyphenations is *Webster's Third New International Dictionary*, Unabridged.

Hyphenation for languages other than U.S. English

Patterns now exist for many languages other than U.S. English, including languages using accented and non-Latin alphabets. CTAN holds an extensive collection of patterns: see `language/hyphenation` and its subdirectories.

A group of volunteers led by Mojca Miklavec and Manuel Pégourié-Gonnard have created a comprehensive package of hyphenation patterns, called `hyph-utf8`; see `http://tug.org/tex-hyphen`.

The list — English words

`as-trologer`	as-trol-o-ger
`as-tronomer`	as-tron-o-mer
`catas-tro-phe`	ca-tas-tro-phe
`catas-trophism`	ca-tas-tro-phism
`chemokine`	chemo-kine
`con-structed`	con-struc-ted
`cy-tokine`	cy-to-kine
`gigan-odes`	giga-nodes
`hip-popota-mus`	hip-po-po-ta-mus
`icono-g-ra-pher`	ico-nog-ra-pher
`icono-graphic`	icon-o-graph-ic
`iconog-ra-phy`	ico-nog-ra-phy
`im-mu-niza-tion`	im-mu-ni-za-tion
`im-munomod-u-la-tory`	im-mu-no-mod-u-la-to-ry
`kilo-n-odes`	kilo-nodes
`leukotriene`	leu-ko-triene
`megan-odes`	mega-nodes
`molec-u-lar`	mo-lec-u-lar
`penalty(ies)`	pen-al-ty(ies)
`pre-dictable`	pre-dict-able
`prefers`	pre-fers
`prostaglandin`	pros-ta-glan-din
`salient`	sa-lient
`ter-a-n-odes`	tera-nodes
`triplex(es)`	tri-plex(-es)
`unin-stan-ti-ated`	un-in-stan-ti-at-ed

Names and non-English words used in English text

`Caratheodory`	Cara-theo-dory

◇ Barbara Beeton
`http://tug.org/TUGboat`
`TUGboat (at) tug dot org`

What does a typical brief for a new typeface look like?

Thomas Phinney

I keep on seeing this question unanswered, and hoping somebody else would answer it, as the answer is . . . long. :)

I'm not sure there is a single "typical brief." One of the things that makes typeface design interesting is the diversity within a seemingly narrow specialty.

As one of my colleagues points out, many of the same questions one considers in designing a new typeface, also apply simply to selecting a typeface, or customizing an existing typeface.

I am a bit unclear on what you mean by "what does it look like"? I am kind of assuming this is a question of content rather than format. If you are looking for formatting advice, please say so. :)

For that matter, the design brief is sometimes never actually written down, nor clearly developed. I do encourage both aspiring type designers, and clients of custom type design, to go through this process and write it all down. I expect that it will be helpful almost all the time, and often immensely helpful, to articulate questions and goals clearly. It sets everyone's expectations and creates reasonable limits.

Many clients won't even know what questions to ask, so the design brief is something that usually gets developed in collaboration between the type designer and the client. Or, if there is no specific client, it can be a matter of asking the questions of oneself, to better focus the design. Being more specific and more seemingly restrictive is likely to result in a more successful work — even if the final fonts are used in ways beyond what the designer originally expected.

A design brief can potentially be a living document, revised over time during the early stages of the project as it is defined. There may be a first round of design brief written in the early exploratory stages, and it may be developed further in one or more additional iterations.

In any case, when taking on a new typeface design, the questions I would be asking the client (or myself) to create the design brief would be these:

1. Who is the client, or target customer?
2. Is it replacing a current typeface? If so, what does the client like and dislike about the current typeface? What is motivating the change?

Figure 1: Fontlab logo.

3. If they considered off-the-shelf options, what did they consider and what did they like about each of them? What did they dislike about each of them? Why did they not go with any of them?

   I found this part incredibly helpful in the process of creating a new logotype recently for my company, Extensis (see fig. 1). We looked at a bunch of specific typefaces and rejected them for a variety of reasons. In the end I took an existing typeface and modified it quite heavily. But I used the knowledge of the strengths and weaknesses of these other typefaces, in terms of what I and my (internal) customers wanted, and that guided what I did to the pre-existing typeface.
4. What is the typeface a vehicle for? What is to be communicated with it? In what way should it flavor the message? Is it intended for a particular project or product?

   In the same case I mention above, we wanted it to feel modern and somewhat techno, yet warm and approachable. Moreover, we had a very playful graphic for the logo — it was almost wacky in how playful it was. We needed to have the type treatment for the logotype be playful enough to not just clash with the graphic, but still a bit more serious, to ground it a bit. It was a careful balancing act.
5. Is there a specific target usage, such as "advertising headlines" or "body text in all publications and online." Even if not. . . What sizes will it be used at? In what media? How will the type be reproduced (imaged, rasterized)? On screen? For web pages? In print?

   Again by way of example, the logo needed to function at pretty small sizes, as logos often do. Some of the typefaces we had entertained were eliminated in part because their weight got too spindly at small sizes on screen; it just wasn't holding up well enough.
6. What else is known about the desired design category?

   In my logo example, we had decided we wanted something in the line of a slab serif typeface, something in a realm defined by typefaces such as Archer, Donnerstag, Vista Slab, and Adelle.

Editor's note: Originally published at `http://www.quora.com/What-does-a-typical-brief-for-a-new-typeface-look-like`. Reprinted with permission.

Figure 2: Some of the Hypatia Sans variants.

My first typeface, Hypatia Sans (see fig. 2), started out with me as a designer coming in thinking I wanted to do a geometric sans serif — but still needing to figure out how to focus it beyond that.

7. How many styles (individual fonts) are desired? Regular, italic, bold and bold italic are four fonts right there (and no, you can't get reasonable quality results by just using algorithmic slanting and bolding). More weights, more widths, or variants intended for different sizes can all add to this total. Families of 8–20 fonts are not unusual today. The largest family I know of is Kepler, comprising 168 (!) fonts.
8. What kind of language coverage is required? Any other particular character set needs (e.g. particular symbols, math capability, whatever). There are a variety of semi-standard character sets and language groupings, but the whole matter is a bit fuzzy around the edges.
9. What kind of typographic extras are required, or might be desirable? For example, these days I consider support for arbitrary fractions and both lining and oldstyle figures (in both tabular and proportional widths) pretty much "basic." Plus at least the five f-ligatures. But other people might think of these as extras. I think of small caps as extra, however, especially with the large language coverage I tend to go for. How about superscript and subscript numbers? A full set of letters for ordinals? Many other possibilities here.

Many of these things essentially multiply together. For example, if you need real small caps, you should probably have them for all the supported languages, and in all the fonts in the family — I would hope it would be (reasonably) assumed, but best to be explicit about it.

◇ Thomas Phinney
http://www.thomasphinney.com

Editor's note: Thomas Phinney is one of the regular teachers of the Crafting Type workshop, an intensive three-day class in type design. It is aimed at anyone with an interest in type and typography, not only professional type designers. It has been given in many cities around the world, and new workshops are actively scheduled. TUG is happy to be able to provide some administrative support for Crafting Type, and we heartily recommend looking into it.

The web site is http://craftingtype.com.

What does a typical brief for a new typeface look like?

Inconsolata unified

Michael Sharpe

Inconsolata is a very fine monospaced font family developed several years ago by Raph Levien, with partial support from TUG, using software of his own design that allowed curve segments to be drawn using *spiros* (a.k.a. Euler spirals or spirals of Cornu) whose defining feature is that curvatures vary linearly with length along the curve. (Such curves have a prominent history in engineering practice, where they were, and are, used for highway and railway curves, and provided the shapes for *French curve* templates which were widely used in engineering drawing before the days of computer-aided design.)

Initial LaTeX support for the font, which was originally provided on CTAN only in Type 1 format, was provided by Karl Berry's `inconsolata` package, where the font had Berry name `fi4`. In 2012, after a bold version became available, I made an enhanced version with Berry name `zi4` that provided a number of new glyphs and lookup tables to allow them to be accessed as alternate forms, along with equivalent LaTeX options for the Type 1 versions. This meant that the CTAN versions of Inconsolata were not the same as those provided by other sources. Recently, there has been cross-platform interest in a reunification.

Though the process is not yet complete, it appears likely that in the near future there will be one master source for Inconsolata, making available the regular and bold weights in two em sizes: 2048 for the TrueType versions and 1000 for the Opentype (`cff`) and Type 1 (`pfb+afm`) versions. In particular, the versions on CTAN will be drawn from this master repository, and future changes in the fonts may require the package maintainer to regenerate the `tfm` files using a script that calls `afm2tfm` with a multitude of encoding files. Of course, changes to glyph names would require making the corresponding changes to the encoding files.

As things stand, the version offered on CTAN and through TeX Live will be updated to the most current version with all features intact. Detailed behavior is in the documentation for the `inconsolata` package, but, to summarize usage in LaTeX:

- regular and bold weights are available;
- options are available to specify:
  - slashed or unslashed zero;
  - upright quotes or original slanted quotes in verbatim text;
  - original lowercase L (el) or a curvier variant.

The last three options are also available in `fontspec` by means of choices of the `StylisticSet`.

Sample LaTeX file fragment:

```
<load text package>
\usepackage[varqu]{inconsolata}
\usepackage{upquote}
...
\begin{document}
\begin{verbatim}
"double-quoted text"
`backticked item`
'single-quoted'
Zero: 0 # note slashed zero as default
# option var0 would use unslashed
...
\end{document}
```

will render as:

```
"double-quoted text"
`backticked item`
'single-quoted'
Zero: 0 # note slashed zero as default
# option var0 would use unslashed
```

Sample XeLaTeX/LuaLaTeX file fragment:

```
\usepackage{fontspec}
\usepackage{libertine}
\setmonofont[StylisticSet=3]{Inconsolata}
% straight quotes
...
\begin{document}
\begin{verbatim}
"double-quoted text"
`backticked item`
'single-quoted'
Zero: 0 # note slashed zero as default
...
\end{document}
```

will render as:

```
"double-quoted text"
`backticked item`
'single-quoted'
Zero: 0 # note slashed zero as default
```

The material above rendered in `Inconsolata` is somewhat larger than the accompanying text, and, in practice, should be `scaled` down to a more appropriate size in the document preamble.

◇ Michael Sharpe
Math Dept, UCSD
La Jolla, CA 92093-0112 USA
msharpe (at) ucsd (dot) edu
http://math.ucsd.edu/~msharpe/

A TUG Postcard or, The Trials of a Letterpress Printer

Peter Wilson

> To become a rich printer you must start as a very rich printer.
>
> Traditional

This is the tale of how and why I came to letterpress print a postcard for TUG.

Background

I have been a LaTeX user since 1985, using it at work to typeset and print a range of technical documents, from reports through code documentation and culminating in several thousand of pages of an ISO International Standard. It is only since I retired that I became involved in traditional letterpress printing. This involves manually selecting pieces of lead type to make up words, sentences, pages and books, putting ink on the type so carefully arranged and pressing a sheet of paper onto the inked type to get a final printed page. You produce quite a few sheets like this, proudly show them to your wife who, after a mere glance, announces that she can see several typos! Back to cleaning up the type, making the necessary replacements and/or additions, and then starting the ink-print-review cycle again.

I got into letterpress work after going on a bookbinding course where one of the participants said that her husband was looking for volunteers to help with traditional printing at the local Highline Community College in a Seattle suburb. He took me on, though I knew nothing about it, gave me perhaps three hours of show and tell and let me get on with it. Most of the shop was devoted to teaching commercial printing; the biggest press that they had was a 4-colour Heidelberg offset press some 30 feet in length, with all text and pictures being set up on a Mac beforehand. The students also had to take a two-day letterpress session in my small corner of the shop. Most were desperate to get back to their Macs but one or two liked the constraints forced by having to use fixed sizes of lead type. There we printed on two Chandler & Price platen presses dating back to the early 20th century. In a platen press the type is put in vertically and if not locked up tightly will scatter itself all over the floor; the students really did not like this as they had to pick it all up, put it back in the correct places in the typecases and start all over again, but you only do it once.

To my surprise I managed to win some awards for my printing, the most prestigious being a Gold Award at the 2009 IAHPC International Gallery of Excellence where there were participants from some 15 different countries. It was for a production called 'Twelve Chinese Great Scholars' done with giclée illustrations and letterpress printing with an accompanying booklet via LaTeX all enclosed in an Oriental style box.

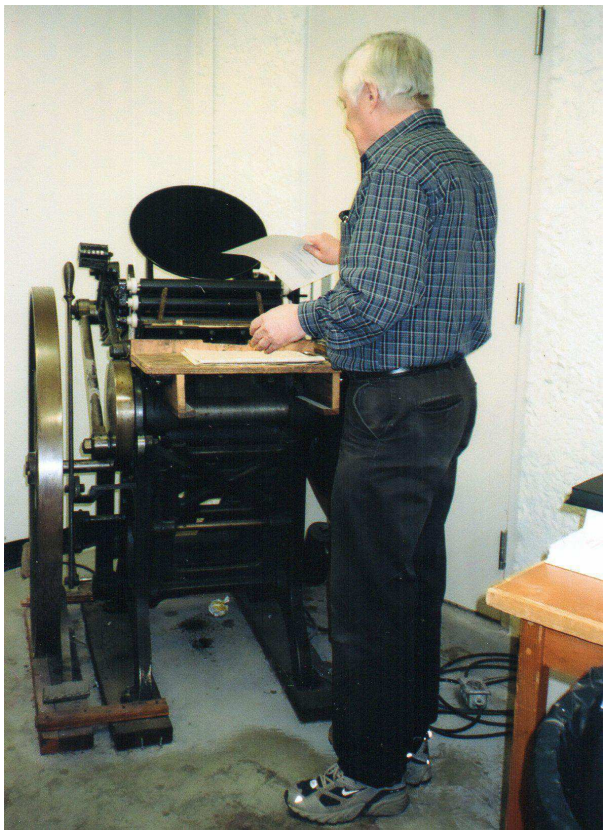


Figure 1: Operating a Chandler & Price Old Style Press

In 2010, after 27 years in the USA, I moved back to the UK and managed eventually to set up a printing and bookbinding workshop in my garage (in the UK it is very rare that a car is kept in a garage).

A partial view of the shop is shown in Figure 2. In the foreground is a nipping press for bookbinding and there are two large type cabinets in the background with each holding twenty typecases and in turn each typecase holds one fount of lead type. There is also a much smaller cabinet holding various founts of brass type for gold tooling the titles on books.

I was fortunate to be able to get a Vandercook SP15 proofing press at a very reasonable price, although it cost nearly as much to have it professionally moved from where it was 200 miles away from my home. This is a horizontal press as opposed to the vertical platen presses which has, for me, the great

Figure 2: Type cabinets and bookbinding equipment

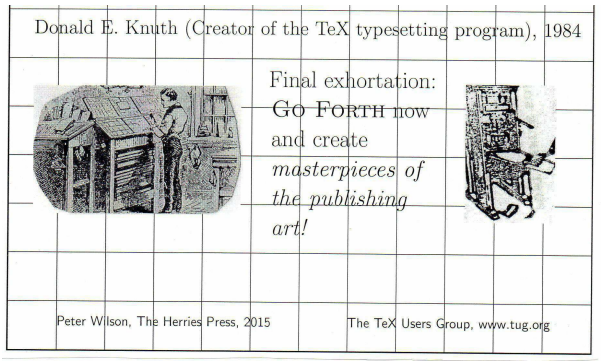


Figure 3: First draft design (in LaTeX)

advantage that once you have the type on the press it will not fall onto the floor. The essential part of this is shown on the next page in Figure 6. It is possible to print sheets as large as 15 by 18 inches (381 by 457 mm), while printing on anything smaller than A5 paper (5.8 by 8.25 inches) tends to be problematic.

The postcard

Karl Berry is very persuasive and in a moment of weakness I agreed to print a few postcards for TUG to be given away as a thank-you for members who persuaded others to become members.[1] I imagined 25 or so but was hooked into producing a hundred. It was left to me to create a design which then Karl, Barbara Beeton and some unnamed reviewers[2] would critique.

The first of my draft layouts, done with LaTeX of course, and shown in Figure 3, consisted of a Knuthian quote flanked by two cuts of a nineteenth century type compositor and of a printing press — these were reproduced using carbon paper, then scanned, and inserted into the LaTeX source as `jpg`'s.[3] The grid pattern was to help me with the final arrangement of the type.

I decided to use Caslon for the printing for two reasons; firstly I am rather fond of it and secondly, and more importantly, I happened to have it in a range of sizes and also in Roman, italic and small caps to match Knuth's original setting. I used 14pt for the Knuth introductory line and 18pt, leaded 6pt, for the quotation itself. I selected 10pt Gill Sans for the imprint at the bottom of the card and the printers flowers were 18pt. I have some 60 founts but none of them span the entire range of sizes and possible styles. Among these are Baskerville, Bembo, Bernhard Cursive, Blado, Caslon, Casteller Titling, Fournier, Gill Sans, Old English, and Times.

But compare this motley selection with Knuth's Computer Modern with the Roman in sizes 5, 6, 7, 8, 9, 10, 12, and 17pt, the bold in 5 to 12pt, small caps in 8, 9, 10, 12 and 17pt, the italic in 8, 9, 10, 12 and 17 pt, which would take up 25 typecases, not to mention the slanted roman, Greek, the typewriter font and others. And then maths fonts are a completely different ballgame! In essence, there is an unlimited supply of characters in digital typesetting. In letterpress, you have to make do with what you have physically got. In my early letterpress printing days I almost completed setting the type for a two page document but ran out of 'h' sorts on the penultimate line of the first page.[4] The text was composed of many sentences like 'Whither art thou going, when, and with whom?' and I had to redesign the whole work to accommodate my limited 'h' supply.

The reviewers asked me to liven up my proposal by adding some sort of decoration to highlight Knuth. There wasn't room for that so I added a top and bottom line of printers flowers. I was also asked to change the lowercase 'e' in 'TeX' to a dropped 'E' as in TEX.

Well, kerning in LaTeX is easy as you just make judicious use of the `\kern` and `\raisebox` command to add or subtract horizontal or vertical space. In

[1] Editor's note: Our membership campaign is detailed at `http://tug.org/membership`; it runs throughout 2015, please take a look!

[2] Editor's note: The other unnamed reviewers were the folks who have volunteered to be on the current incarnation of the TUG membership committee: Kaja Christiansen, Jim Hefferon, Klaus Höppner, Robin Laakso, Steve Peter, Boris Veytsman, and Dave Walden. Peter was remarkably patient and gracious in the face of our ignorant ideas!

[3] 'Cut' is a somewhat generic term for an engraved illustration of some kind mounted on a wooden block to bring it up to typeheight so that it can be printed along with the regular type.

[4] 'Sort' is the generic term for a single piece of lead type.

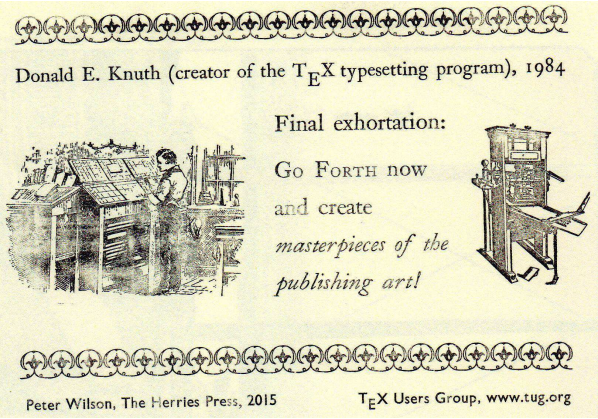


Figure 4: Second draft printed design

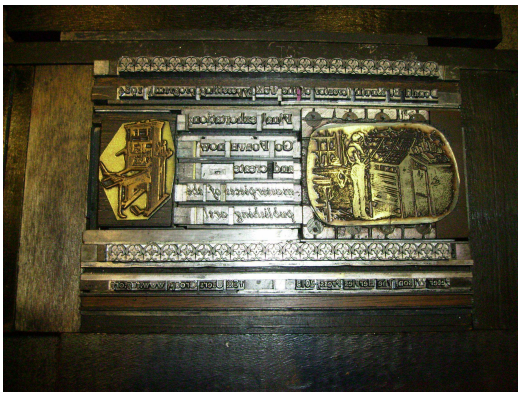


Figure 5: The final type locked up in the forme

Figure 6: The forme on the press

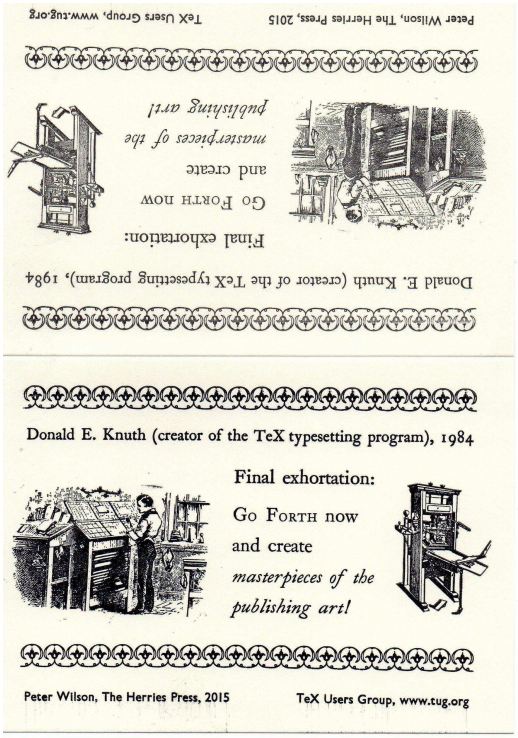


Figure 7: Final printing

letterpress printing when you are dealing with fixed sizes of lead type it is not so simple. For a positive horizontal kern you can add some extra spacing between the adjacent sorts, ranging from a slip of cigarette paper through copper and brass shims to lead spacers. Any other kerning requires trimming pieces of lead, either above or below or before a sort; not easy. However, I did my best to meet their requests with the result as in Figure 4 (the card got a bit askew when I scanned it originally and I have since disposed of the original).

I was not happy with my attempts to kern the 'E' and fortunately the reviewers were of the same opinion so I reverted back to the regular 'e'. I then spent some time making minor adjustments to the positions of all the elements on the card — a process of adding and subtracting bits of lead until it all looked right. The final positioning of the type, cuts, and spacers in the forme is shown in Figure 5. The forme consists of a rectangular cast-iron frame, called a chase, enclosing the type to be printed, which must be firmly locked up in the chase so that the whole lot can be picked up and moved around without anything falling out.

A US postcard is about the same size as an A6 sheet, and as I mentioned, printing for me on anything smaller than A5 is not so easy. As it happened I couldn't find any suitable A6 paper so I used A5 instead. As I had only one copy of each of the cuts I could only print one postcard at a time, so I printed on one half of the A5, waited two or more days for the ink to dry and then printed on the other half of the sheets, one of which is shown in Figure 7.

I was limited in the amount of drying space that I had, about enough for twenty-five sheets, so the whole process became somewhat protracted.

For one reason or another I managed a high failure rate of about 10%, which just lengthened the

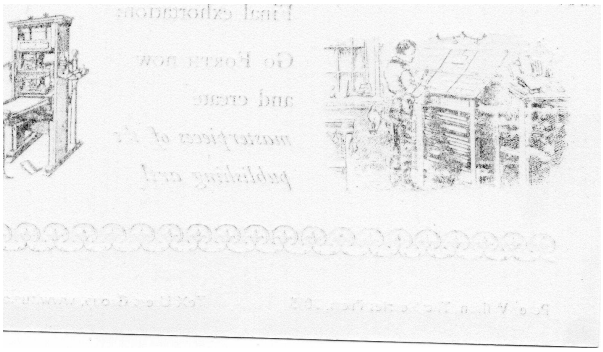


Figure 8: Ink on the back

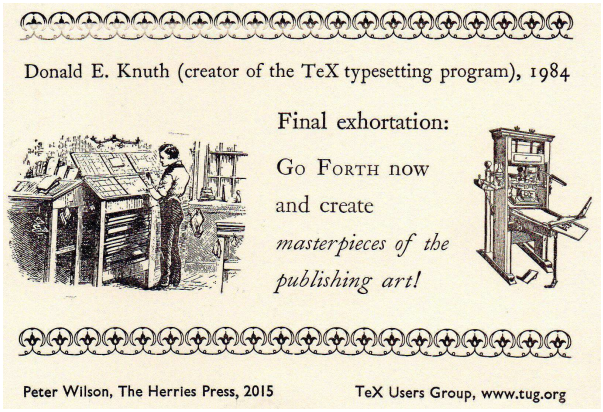


Figure 9: Missing ink

time. Apart from the usual occasional smudging of ink when taking a sheet out of the press and the odd misfeed of the paper I had two major problems. The first is illustrated in Figure 8 where somehow I managed to get ink on both the front and the back of the cards. A good cleaning of everything solved that one.

The more troublesome difficulty is shown in Figure 9 where there is not enough ink on the top row of printers flowers. I spent a long time putting slips

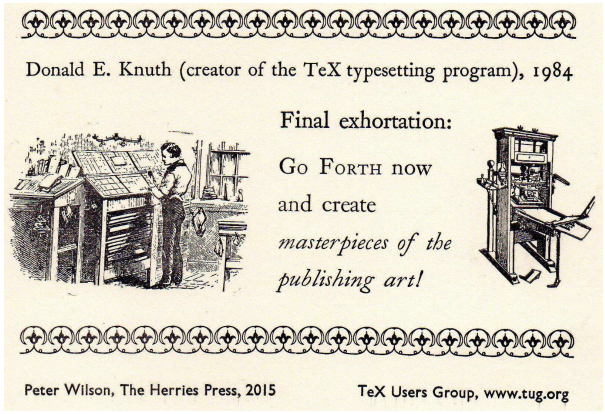


Figure 10: A postcard in its deliverable state

of paper under the sorts that were not fully printing to raise them up so that the ink rollers would be sure to touch them. This didn't work. Eventually I found that I was moving the ink rollers too fast across the type and they were partially bouncing over the top row of type. I reduced my printing rate from about four to three per minute and all seemed well after that.

The end

Perhaps the surprising thing after all this is that I did manage to produce the hundred postcards that Karl requested. A deliverable card is shown in Figure 10. And then, almost to add insult to injury, Karl asked me to write something up about the whole process for *TUGboat*, hence this rather long-winded piece — just don't blame me.

◇ Peter Wilson
12 Sovereign Close
Kenilworth, CV8 1SQ
UK
herries dot press (at)
earthlink dot net

Typographers' Inn

Peter Flynn

1 Portable typesetting

I recently had to reinstall TEX Live for a novice user after she decided to replace a stolen Macbook with a generic unbranded laptop PC. The abruptness of this change of platform is not something TEX users would normally worry about because TEX works perfectly happily pretty much everywhere, but this user's main concern was that she lost her iPhone as well, and was making a parallel switch to Android... and had heard that this would enable her to have LATEX on the phone as well.

I had encountered *VerbTEX*, which is an Android editor that submits a document to a web service which does the typesetting. It's a great system, as it does away with the need to have all of TEX installed, but if you are stranded without a connection, it's impossible to carry on working on a document that you want to be able to preview.

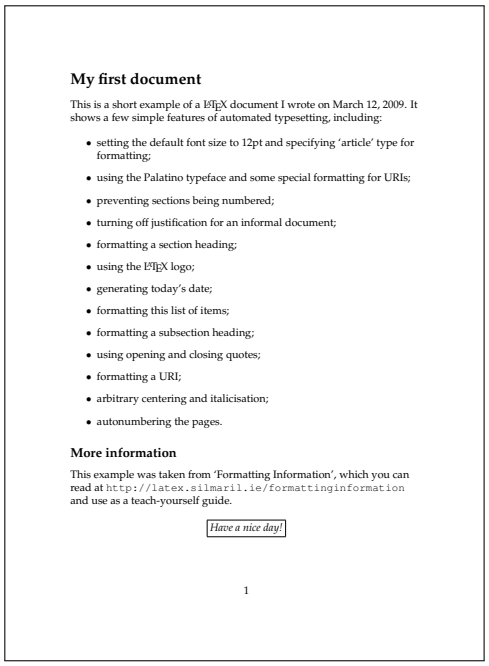
Enter Jiří Marek's *LATEX Editor*. Despite the name, this is a fully-fledged locally-executing LATEX system (although of course you do need a connection to install it and download any additional packages). My first thought was to try the `quickstart.tex` document I use in the LATEX course I teach — the same one as in *Formatting Information*.[1] It worked first time, and as I have *Kingsoft Office* (which includes PDF preview) and the *PrinterShare* app, I could even print it. Full marks.

Next up was the draft of this article, which I was somewhat sceptical about, as ltugboat.cls is pretty complex, and has to deal with a lot of unusual formatting. *LATEX Editor* handled it entirely correctly, downloading all the packages needed on-the-fly. There was a moment when I thought it had hung, but in fact it was just downloading packages: the lack of a progress meter does make this a little unnerving the first few times until you get to trust it.

I don't have an iPhone or an iPad, so I can't compare this with whatever facilities are available in that ecosystem, and I am admittedly using a Galaxy Note 4, which is a phone masquerading as a tablet, with an external Bluetooth keyboard — I haven't tried this setup on anything like the really tiny Android phones.

However, it's clear that developing packages and classes, and experimenting with typographic solutions can continue unabated on a pocket device, with no serious technological barrier. I haven't yet found out how to import my personal PS fonts with their associated maps and font definitions, so explicitly typographical development is restricted to what is available as packages from CTAN (which is substantial). The current version of *LATEX Editor* is running *pdflatex* from TEX Live 2012, not XELATEX, so there

[1] `http://latex.silmaril.ie/formattinginformation/quickstart.html#quickstart`



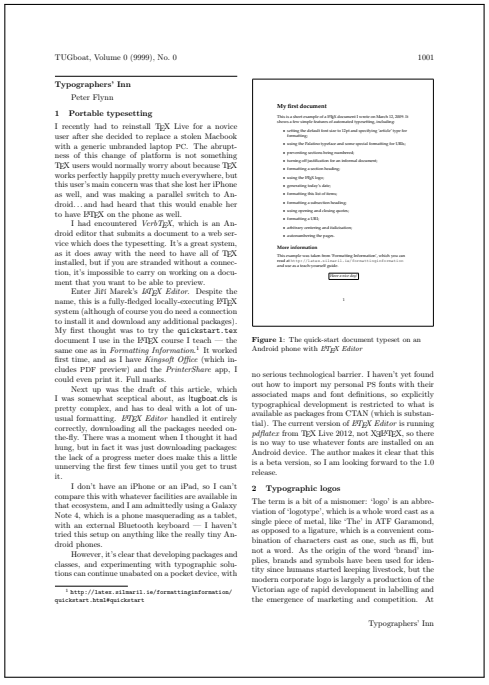


Figure 1: The quick-start document (top) and this article (bottom) typeset on an Android phone with *LATEX Editor*

is no way to use whatever fonts are installed on an Android device. The author says this is a beta version, so I am looking forward to the 1.0 release.

2 Typographic logos

The term is a bit of a misnomer: 'logo' is an abbreviation of 'logotype', which is a whole word cast as a single piece of metal, like 'The' in ATF Garamond, as opposed to a ligature, which is a combination of characters cast as one, such as ffi, but not a word.

Brands and symbols have been used for identity since humans started keeping livestock (branding!), but the modern corporate logo is largely a production of the Victorian age of rapid development in printed labelling combined with the emergence of marketing and competition. At the time, however, logos were more a convenient way of flagging your products, rather like the exhortations to look on the jar; 'none genuine without my signature!'

The development of colour lithography meant logos could be hand-drawn as part of a larger image, and still reproduced in bulk, whereas in letterpress, anything other than combinations of type would mean making a block. Type-only logos are still with us, from IBM to TEX, and can be surprisingly difficult to construct even (like IBM or METAFONT) when they are simply letters in a given font.

Using them in the text is frowned upon, typographically speaking (TEX must surely be the worst offender here). Charles Fyffe, in his book on copyfitting [1] (now dated but still a mine of useful information) says:

> Don't use the client's name-style in the copy and expect it to be read, unless his name-style is in a type and you are using it for the body copy [Phew! that lets TEX out — PF]. This is especially true of a name-block (plate) with white letters out of black.
>
> The client, on the other hand, tries to use his name-style everywhere — I have even known one who insisted that minute name-blocks be inserted in the copy...

What he's referring to is the practice, still occasionally seen, of using *your logo* for every mention of *your product* or *company name* in the text, especially of an advertisement, such as this one slated in the blog of one Desmond Tan [5] and reproduced without permission in Figure 2. The practice is also condemned on the *Typophile* blog [3], where 'DO NOT' and 'NEVER' feature strongly.

The Honeywell company has an explicit rule in their instructions to designers [2]: 'In body copy or text, do not use the Honeywell logo. Portray the word Honeywell in the same font as your body copy/text.' They even provide an example (Figure 3). See also *The TEXbook* [4, ch. 1, para. 4] on the distinction between Honeywell's TEX and TEX.

Figure 2: Coke advert showing abuse of the logo in the text

Figure 3: Honeywell's example of how not to do it

Afterthought

Perhaps the most extreme form of embedding is that done by a recent instance of actual OpenType font code, for which I make no apology for posting the naked URI: `http://pixelambacht.nl/2015/sans-bullshit-sans/`

References

[1] Charles Fyffe. *Basic Copyfitting*. Studio Vista, London, 1969.

[2] Honeywell Security. *Using the Honeywell Logo*. Honeywell International Inc, Morristown, NJ, 2015. `http://www.security.honeywell.com/resources/branding/logo/`.

[3] joshuaone9. font as logo vs used in body copy. *Typophile*, Feb 2008. `http://typophile.com/node/42144`.

[4] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading, MA, Jun 1986.

[5] Desmond Tan. Something Interesting. `https://desmondtan91.wordpress.com/`.

◇ Peter Flynn
Textual Therapy Division,
Silmaril Consultants
Cork, Ireland
`peter (at) silmaril dot ie`
`http://blogs.silmaril.ie/peter`

LaTeX News

Issue 21, May 2014

Contents

Scheduled LaTeX bug-fix release

This issue of LaTeX News marks the second bug-fix release of LaTeX 2ε (standard LaTeX) since shifting to a new build system in 2009. Provided sufficient changes are made, we expect to make such releases yearly or every two years, in sync with TeX Live.

Release notes

This release makes no changes to the core code in the LaTeX 2ε format but there are a small number of documentation fixes (not listed here). In addition several packages in the base and required areas have been updated as detailed below.

This has been done in accordance with the philosophy of minimising problems in both forwards and backwards compatibility, so most of these changes should not be noticed by the regular LaTeX user.

References in the text below of the form "graphics/3873" are to bug reports listed at:
`http://latex-project.org/cgi-bin/ltxbugs2html`

fixltx2e updates

There are a number of bugs and faulty design decisions in LaTeX 2ε that should have been corrected long ago in the kernel code. However, such corrections cannot be done as this would break backwards compatibility in the following sense. A large number of documents exist by now that have worked around the bug or have even made use of a particular misfeature. Thus changing the kernel code would break too many existing documents.

The corrections for these types of bug have therefore been collected together in a package that can be loaded only when needed; its name is fixltx2e. For this release we made the following changes to this package:

- Misspelled float placement specifiers such as `\begin{figure}[tv]` instead of `tb` are silently ignored by the kernel code. Now we test for such letters and issue an error message.
- LaTeX's float handling algorithm can get out of sync if you mix single and double-column floats (as they are placed independently of each other). This was corrected in fixltx2e a few years ago but the fix was not perfect as one situation using `\enlargethispage` generated a low-level TeX error. This behaviour of the package is now improved.

New fltrace package

For years the file `ltoutput.dtx` contained some hidden code to trace the detailed behaviour of the float placement algorithm of LaTeX. Prompted by questions on StackExchange we now extract this code into a new fltrace package. To see the float algorithm in action (or to understand why it decides to place all your floats at the very end of the document) use

```
\usepackage{fltrace} \tracefloats
```

To stop tracing somewhere in the document use `\tracefloatsoff` and to see the current value of various float parameters use `\tracefloatvals`. As the package is identical to the kernel code with tracing added, it may or may not work if you load any other package that manipulates that part of the kernel code. In such a case your best bet is to load fltrace first.

inputenc package updates

The inputenc package allows different input encodings for LaTeX documents to be specified including the important `utf8` option used to specify the Unicode UTF-8 encoding. A common mistake in documents has been to also include this option when using the Unicode-based TeX engines LuaTeX and XeTeX producing strange errors as these engines natively deal with UTF-8 characters.

If a document stored in an 8bit encoding is processed by pdfTEX, it needs the inputenc package to work correctly. However, if such a document is processed unchanged by LuaTEX or XETEX, then accented characters may silently get dropped from the output.

The package has been modified so that if used with LuaTEX or XETEX, then it just issues a warning if `utf8` or `ascii` is specified, and stops with an error for any other encoding requested.

One further improvement has been made to the encoding definition files (`.def`) used by inputenc: the catcode of `@` is now saved and restored when reading them instead of always using `\makeatother` inside the files (latex/4192).

The tools directory

In the past each of the sub-directories in the "required" section of the LATEX distribution contained a single `.ins` file to generate the code files from the source files. We have now started to provide individual `.ins` files for each of those packages that are likely to require updates outside a major LATEX release.

multicol updates

Version 1.8 of multicol implements some improvements/fixes and one extension. In the past the balancing algorithm enlarged the column height until it found a solution that satisfied all constraints. If there were insufficient break points then the final column height could have been much larger than expected and if that happened near the end of the page it resulted in the text overflowing into the bottom margin. This situation is now detected and in that case a normal page is cut and balancing is resumed on the next page. Some overflow is still allowed and controlled via the parameter `\maxbalancingoverflow`.

The use of `\enlargethispage` is now properly supported within the environment. Finally a new command `\docolaction` was added to allow the execution of code depending on the column in which the command is executed. See the documentation for details.

Bug fixes: the new version fixes both a color leak that could happen in certain situations and the problem that `multicols` could mess up the positioning of `\marginpar`s that followed the environment.

tabularx updates

The restrictions on embedding `\tabularx` `\endtabularx` into the definition of a new environment have been relaxed slightly. See the package documentation for details.

showkeys updates

The showkeys package has been updated to fix problems if used at the start of list items, and to work if brace groups (`{` and `}`) are used in the optional argument of `\cite`. (tools/4162, tools/4173)

color updates

The `\nopagecolor` command suggested by Heiko Oberdiek, available for some years in the pdftex option, has been added to the core package as suggested in graphics/3873. Currently this is supported in the driver files for dvips and pdftex. Patches to support other drivers are welcome.

graphicx updates

The graphicx version of `\rotatebox` now allows `\par` (and blank lines) in values, to match the change made to the graphics version some years ago. See graphics/4296.

keyval updates

All parsing used in the keyval package has been changed to allow `\par` (and blank lines) in values. (A second change, to parsing of brace groups in a construct such as `key={{{value}}}`, was reverted in v1.15.) See graphics/3446.

Standard LATEX (LATEX 2ε) and expl3

The substantial collection of innovative code in expl3 implements a new programming language that has for a while now been used by some writers of LATEX 2ε packages. This code has recently also been made available for use on top of plain TEX or ConTEXt, largely to support generic packages that are supposed to work with different flavours of TEX. These uses in no way affect authors of LATEX documents and such LATEX 2ε packages will continue to work as advertised by their authors with standard LATEX.

This code base will also become an important foundation for the kernel of LATEX3 and so the new programming language can be described as 'The LATEX3 Programming Language'. However, if you see or hear that a package 'uses LATEX3' then it remains very unlikely (as yet) to mean that the package is part of some 'new version of LATEX'.

News about the development and use of expl3 and about other developments in the LATEX3 code base is reported regularly in the LATEX3 News series (`http://latex-project.org/l3news/`), the most recent issue of which was published in March 2014.

Beamer overlays beyond the \visible

Joseph Wright

Last year, I looked at the beamer overlay concept with relative slide specifications to produce dynamic slide structures (Wright, 2014). Prompted by a question on TEX Stack Exchange ('Tarass', 2014), here I'm going to look at a related area: action keywords.

The 'standard' beamer overlay system does the same thing as the \visible command: makes things appear and disappear, but always keeps space for them on the slide. However, beamer also provides \only, which completely omits items not visible on a slide. So the question was how to combine this idea with the general overlay concept.

It turns out that this is quite straightforward if you know what to look for. The standard beamer overlay syntax, for example

```
\item<+->
```

supports the inclusion of an action type to specify what the overlay should do. That action is given as a keyword and an @ before the overlay number(s). So, for example

```
\begin{itemize}
  \item First item
  \item<only@1> Second item
  \item<only@2> Replacement second item
...
```

will show Second item on the first slide and then replace it entirely with Replacement second item on the second slide. That approach can be combined with the idea of relative slide specifications, as I talked about before, to give something like

```
\documentclass{beamer}
\begin{document}
   \begin{frame}
   \begin{itemize}[<+->]
      \item item 1
      \item item 2
      \item<only@+-.(2)> item 3
      \item item 4
      \item item 5
   \end{itemize}

   \end{frame}
\end{document}
```

to have the 'normal' items appear one at a time but with item 3 only on slides 3 and 4.

This doesn't just apply to only: other keywords that work here include visible and alert. The latter tends to be seen with another syntax element; namely, | to separate out appearance from a second action. A classic example of that is

```
\documentclass{beamer}
\begin{document}
   \begin{frame}
   \begin{itemize}[<+->]
      \item item 1
      \item item 2
      \item<+-|alert@+(1)> item 3
      \item item 4
      \item item 5
   \end{itemize}
   \end{frame}
\end{document}
```

where item 3 appears on the third slide and is highlighted on the fourth one. (Note that both + substitutions in this line use the same value for the pause counter, hence needing the (1) offset.) That's useful even without the 'one at a time' effect, for example

```
\documentclass{beamer}
\begin{document}
   \begin{frame}
   \begin{itemize}
      \item item 1
      \item item 2
      \item<alert@+(1)> item 3
      \item item 4
      \item item 5
   \end{itemize}
   \end{frame}
\end{document}
```

highlighting the item on the second slide.

A bit of imagination with this syntax can cover almost any appearance/disappearance/highlight requirement. As I said before: the key thing is not to overdo it!

References

'Tarass'. "Beamer: including items only some slides using a relative syntax". tex.stackexchange.com/q/205625, 2014.

Wright, Joseph. "The beamer class: Controlling overlays". *TUGboat* **35**, 31–33, 2014. tug.org/TUGboat/35-1/tb109wright.pdf.

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

Glisterings: Here or there; Parallel texts; Abort the compilation

Peter Wilson

> A stately rocke beset with Diamonds faire,
> And pouldred round about with Rubles red,
> Where Emeralds greene doo glister in the air,
> With Mantill blew of Saphyres ouer spred.
>
> *The Ship of safegarde*, Barnabe Googe

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two while hopefully not making things worse through any errors of mine.

> In a civil war, a general must know — and I'm afraid it's a thing rather of instinct than of practice — he must know exactly when to move over to the other side.
>
> *Not a Drum was Heard: The War Memoirs of General Gland*, (unpublished radio play, 1959) Henry Reed

1 Here or there

Paul Kaletta asked on `ctt` [slightly edited]:

I am writing a twoside document which means that even and odd pages have different margins. Unfortunately all images I include are aligned with the left side of the text on every page. Some of them are broader than the line width and protrude into the right margin, which is nice for odd pages, but looks weird for even ones.

I would love to align the images to the inner margin, so that they always protrude to the outer one. Is this possible?

Heiko Oberdiek gave a solution so that an image would not exceed the width of the text plus the marginpar area [3].

This has been a problem that has cropped up from time to time on `ctt`. More generally the problem is how to decide into which margin something should be put, and then put it there. The code below for the first problem is based on code that I wrote for my memoir class. This version requires the changepage package [8] for correctly deciding whether an odd or even page is being typeset.[1]

The `\pikmargin` workhorse macro, used for specifying a margin, takes one argument which must be one of: `left`, `right`, `outer`, or `inner`. The result is `\pkmarg` which is in the range 0–3 for the allowed arguments, otherwise it is −1. The code is rather tedious.

```
\usepackage{changepage}
\newcommand*{\pikmargin}[1]{\bgroup
  \def\targ{#1}\def\parg{left}%
  \ifx\targ\parg
    \gdef\pkmarg{0}%
  \else
    \def\parg{right}%
    \ifx\targ\parg
      \gdef\pkmarg{1}%
    \else
      \def\parg{outer}%
      \ifx\targ\parg
        \gdef\pkmarg{2}%
      \else
        \def\parg{inner}%
        \ifx\targ\parg
          \gdef\pkmarg{3}%
        \else
          \gdef\pkmarg{-1}%
        \fi
      \fi
    \fi
  \fi
\egroup}
```

The `\settheside` workhorse macro takes one argument, the value of `\pkmarg` from `\pikmargin`, and sets `\ifputatright` TRUE or FALSE according to whether material should be put into the right or left margin. The basic algorithm is:

1. A negative argument is converted to 2 (outer).
2. For two columns always the nearest margin.
3. For one sided documents:
   0 (left) FALSE
   not 0 (all else) TRUE
4. For two sided documents:
   0 (left) FALSE
   1 (right) TRUE
   2 (outer) TRUE on an odd page and FALSE on an even page
   3 (inner) FALSE on an odd page and TRUE on an even page

The code is tedious, even more so than for the previous macro.

```
\newif\ifputatright
\makeatletter
\newcommand*{\settheside}[1]{%
  \def\m@rgcode{#1}%
  \ifnum #1<0\relax
    %% error! write message and set to 'outer'
  \typeout{Error! arg is '#1'. Set to 'outer'}
    \def\m@rgcode{2}%
  \fi
```

[1] Because of the asynchronous nature of TEX's page breaking algorithm simply checking the page number does not always lead to the correct result. The changepage macros are an integral part of memoir.

```
  \if@twocolumn
    \if@firstcolumn
      \putatrightfalse
    \else
      \putatrighttrue
    \fi
  \else
    \checkoddpage%  from the changepage package
    \if@twoside
      \ifcase\m@rgcode\relax
        \putatrightfalse
      \or%    1 = left
        \putatrighttrue
      \or%    2 = outer
        \ifoddpage
          \putatrighttrue
        \else
          \putatrightfalse
        \fi
      \or%   3 = inner
        \ifoddpage
          \putatrightfalse
        \else
          \putatrighttrue
        \fi
      \fi
    \else%   1-sided
      \ifnum\m@rgcode=0\relax
        \putatrightfalse
      \else
        \putatrighttrue
      \fi
    \fi
  \fi}
\makeatother
```

You can use the `\pikmargin` and `\settheside` macros directly but in case there might be more than one kind of material to be put into the margins it is better to be conservative and use them indirectly.

With the two workhorse macros in hand, here is code for letting overwide images extend a particular distance, `\ximwidth`, into the margin.

`\pikimagemargin` is for selecting the margin for a wide image. The margin code is stored as `\pkimg`.

```
% \usepackage{graphicx} need this package
\newcommand*{\pikimagemargin}[1]{%
  \pikmargin{#1}%
  \ifnum \pkmarg<0\relax
    %% error! write message and set to 'outer'
    %% or perhaps to something more appropriate
  \typeout{Error! arg is '#1'. Set to 'outer'}
    \def\pkimg{2}%
  \else
    \let\pkimg\pkmarg
  \fi}
```

The next bit of code sets the maximum width for an image.

```
\newdimen\ximwidth% extra width
\newdimen\maximwidth% max total width
\makeatletter
\newcommand*{\maxiw}{%   MAX Image Width
  \ifdim\Gin@nat@width>\maximwidth
    \maximwidth
  \else
    \Gin@nat@width
  \fi}
\makeatother
```

An external image is included by calling `\MaxImage` which is a wrapper around the regular graphicx package `\includegraphics` macro and takes the same arguments, except for the optional width argument which is supplied internally.

```
\newcommand*{\MaxImage}[2][]{%
  \par\noindent
  \settheside{\pkimg}%
  \ifputatright
  \else
    \hspace{0pt minus \ximwidth}% move left
  \fi
  \includegraphics[{#1,width=\maxiw}]{#2}%
  \ifputatright
    \hspace{0pt minus \ximwidth}%
  \fi
  \par}
```

The general user scheme is:

```
%% set the dimensions
\setlength{\maximwidth}{\textwidth}
\setlength{\ximwidth}{\marginparwidth}
\addtolength{\maximwidth}{\ximwidth}
%% specify the margin (say the outer)
\pikimagemargin{outer}
...
%% image may be in a figure, but need not be
\begin{figure}
\centering
\MaxImage[height=\textheight,
          keepaspectratio]{myimage}
\caption{...}
\end{figure}
```

> To do just the opposite is also a form of imitation.
>
> *Aphorismen*, Georg Christoph Lichtenberg

2 Parallel texts

2.1 Opposites

On occasion somebody wants to set two documents in parallel on facing pages. This is typically in the form of an original in one language on even numbered pages and a translation in another language on the facing odd numbered pages. The ledpar package [10]

is designed for this purpose, enabling individual line numbering and multiple footnotes on the parallel pages. But sometimes this may be overkill. Stephen Hicks [2] presented a method in response to a query on `texhax`, where it didn't matter if one of the texts was much longer than the other (if necessary the shorter text being 'completed' with blank pages). He explained his basic algorithm as:

1. *Load both documents into separate boxes (i.e., galleys)*
   ```
   \setbox\left@box\vbox\bgroup
       \input left\egroup
   \setbox\right@box\vbox\bgroup
       \input right\egroup
   ```
   This might lead to difficulties if anything in the documents have, say, `\eject` *or anything else weird re: page handling, or it might just work if the whatsits behave well inside boxes.*
2. *Alternately* `\vsplit` *off* `\textheight` *from each box and* `\unvbox` *it into the current page, followed by a* `\clearpage`.

Stephen's code for implementing this was as follows, except that I have made a minor change described later, and exercised some editorial privilege.

```
\documentclass{report}% or other class
...
\makeatletter
\newbox\left@box \newbox\right@box
\newenvironment{leftpage}{%
  \global\setbox\left@box\vbox\bgroup}%
  {\egroup}
\newenvironment{rightpage}{%
  \global\setbox\right@box\vbox\bgroup}%
  {\egroup}
\def\alternate{%\cleardoublepage
  \cleartostart
  \let\@next\@alternate
  \ifdim\ht\left@box=\z@\ifdim\ht\right@box=\z@
    \let\@next\relax\fi\fi
  \@next}
\def\@unvsplit#1{\ifdim\ht#1=\z@\vbox{}\else
  \setbox\z@\vsplit#1 to\textheight\unvbox\z@
  \fi}
\def\@alternate{\@unvsplit\left@box\eject
  \@unvsplit\right@box\eject\alternate}
\makeatother
...
\begin{document} ...
\begin{leftpage}
\input{lefttext}
\end{leftpage}
\begin{rightpage}
\input{righttext}
\end{rightpage}
\alternate
... \end{document}
```

As Stephen said, there are limits to what can be successfully included in the parallel texts. For example, footnotes may throw things out of kilter and page headings can get out of synch if they are changed inside either of the texts by, say, including some `\section`s.

The technical change I made was replacing the macro `\cleardoublepage` with the new one named `\cleartostart`. This is called just before the left–right printing starts. With `\cleardoublepage` the left text starts on an odd page and continues on odd pages while the right text then starts on the following even page. It seems more logical to me that the left text should start on an even numbered page, this being the left of a two page spread. The standard `\clearpage` moves to the next page, which may be odd or even, while the `\cleardoublepage` moves to the next odd page. The `\cleartostart` macro, which is based on `\cleartoevenpage` from the memoir class [9], moves to the next even page.

```
\newcommand*{\cleartostart}{\clearpage
  \ifodd\c@page\hbox{}\newpage\fi}
```

2.2 Equals

Thomas Thurman, who described himself as a poet and programmer, posted to `ctt` saying [6]:

I have a particular typesetting task, described below. Can you tell me whether it's possible in TeX without major upheaval? (Pointers as to how it's possible are welcomed, but at the moment I want to check that it's possible at all.)

I have two source documents P and Q. P consists (as you might expect) of a series of words separated by spaces and punctuation. Q consists of exactly the same number of entirely different words, but separated by the same punctuation. The words may not necessarily be the same length, but there will be the same number of them.

So P might run "I am (of course) shocked! and appalled!" and Q might run "We drink (in summer) lemonade! and Pimms!"

What I want to do is to turn P and Q into a TeX document that either:

- consists of two columns per page, the left from P and the right from Q, but on each line the number of words in each column is the same. (So if there are five words from the P column on the first line, there are five words from the Q column on the first line.)

or

- consists of pages alternately from P and Q, but for each line the number of words on that line is equal to the number of words on the same line on the facing page.

Either is a good solution. (Both would be wonderful.)

Of course if P has a run of long words then the matching Q line will contain a lot of whitespace. This is quite all right.

This resulted in a conversation between Bruno Le Floch and Jean-François Burnol ending with essentially the following code from Jean-François [1] (I have edited it slightly to better fit the two-column format). I can't explain how it works any better than what you see.

```
\makeatletter
% ======== Some helper macros
\let\xpf\expandafter
\def\addtobuff#1#2{\xpf\def\xpf#1%
                \xpf{#1 #2}}
\long\def\ifneitherempty#1#2{%
  \xpf\ifx\xpf a\detokenize{#1}a%
    \xpf\@gobble
  \else
    \xpf\ifx\xpf a\detokenize{#2}a%
      \xpf\xpf\xpf\@gobble
    \else
      \xpf\xpf\xpf\@firstofone
    \fi
  \fi}

% ======== Splitting into paragraphs

\long\def\longsbs #1#2{%
   \longsbs@aux #1\par\Q #2\par\Q}

\long\def%
\longsbs@aux #1\par#2\Q #3\par#4\Q{%
  \sidebyside{#1}{#3}% do one paragraph
  \bigskip % space between paragraphs
  % If either is empty, we're done
  % else do "\sidebyside"
  \ifneitherempty{#2}{#4}%
  {\longsbs@aux #2\Q #4\Q
  }}

% ======== Splitting at each space

\def\sbs@parse #1 #2 \Q #3 #4 \Q{%
  \sbs@step{#1}{#3}%
  % if either text is empty,
  % we are (almost) done
  % else continue
  \ifneitherempty{#2}{#4}%
  {\sbs@parse #2 \Q #4 \Q}}

% ======= Checking the size of each line
% ======= and printing it when it's ready

\newif\ifsbs@break
\def\sbs@step#1#2{%
    \setbox1=\hbox{\sbs@buffi{} #1}%
    \setbox2=\hbox{\sbs@buffii{} #2}%
    \ifdim\wd1>.4\hsize\sbs@breaktrue\else
    \ifdim\wd2>.4\hsize\sbs@breaktrue\else
    \sbs@breakfalse\fi\fi
    \ifsbs@break\sbs@writeline%
      \def\sbs@buffi{#1}%
      \def\sbs@buffii{#2}%
    \else
      \addtobuff\sbs@buffi{#1}%
      \addtobuff\sbs@buffii{#2}%
    \fi}

\def\sbs@writeline{%
    \hbox to \hsize{\hss%
         \hbox to .4\hsize{\pr@buffi}%
         \hskip.1\hsize%
         \hbox to .4\hsize{\pr@buffii}%
     \hss}}

% ========= Master function

\def\sidebyside#1#2{%
    \def\sbs@buffi{\noindent}%
    \def\sbs@buffii{\noindent}%
    \sbs@parse #1 \Q #2 \Q
    \sbs@writeline% flush the last line
}
```

I have added the following code so that the user can specify if the left and right texts are to be set flush left (`[l]`), centered (the default) or flush right (`[r]`).

```
\newcommand*{\setsbsleft}[1][c]{%
  \def\pr@buffi{\hfill\sbs@buffi\hfill}%
\def\@tempa{#1}\def\@tempb{l}
  \ifx\@tempb\@tempa
    \def\pr@buffi{\sbs@buffi\hfill}%
  \else
    \def\@tempb{r}%
    \ifx\@tempb\@tempa
      \def\pr@buffi{\hfill\sbs@buffi}%
    \fi
  \fi}
\newcommand*{\setsbsright}[1][c]{%
  \def\pr@buffii{\hfill\sbs@buffii\hfill}%
\def\@tempa{#1}\def\@tempb{l}
  \ifx\@tempb\@tempa
    \def\pr@buffii{\sbs@buffii\hfill}%
  \else
    \def\@tempb{r}%
    \ifx\@tempb\@tempa
      \def\pr@buffii{\hfill\sbs@buffii}%
    \fi
  \fi}

%% center the texts
\setsbsleft
\setsbsright
\makeatother
```

The following is a short example of using the `\longsbs` macro which, unfortunately, may have difficulties if either of its arguments includes any macros. In this case the texts are set flush right and flush left.

```
\setsbsleft[r]
\setsbsright[l]
\longsbs {%
    I am (of course) ...

    Can you tell ...
}{%
    We drink (in summer) ...

    P consists ...
}
```

I am (of course) shocked! and appalled! I have a particular typesetting task, described herein.

We drink (in summer) lemonade! and Pimms! I have two source documents P and Q.

Can you tell me whether it's possible in TeX . . . at all.

P consists (as you might expect) of a series . . . same punctuation.

> Eternity's a terrible thought. I mean, where's it all going to end?
>
> *Rosencrantz and Guildenstern are Dead*, Tom Stoppard

3 Abort the compilation

Rasmus Villemoes wrote to `ctt` [7]:

I have a document which is only meant to be typeset using pdflatex. It is rather large, and the first pdf-only stuff doesn't occur until quite late. So if one accidentally compiles with latex it takes a couple of minutes before the error is discovered. I would therefore like to insert some code shortly after `\documentclass` *which aborts the compilation with an error message unless running under pdflatex.*

Both Lars Madsen and Heiko Oberdiek replied and the following code is a merge and extension of their responses. The definition of `\abort` is from Heiko and following a comment by Lars I included using the ifxetex package [5] in addition to the originally suggested ifpdf package [4] as both `pdflatex` and `xelatex` generate pdf output.

```
\documentclass[...]{...}
\usepackage{ifpdf}
\usepackage{ifxetex}
\newcommand*{\abort}{}
\ifpdf\else
  \ifxetex\else
    \typeout{You must be in PDF mode.
      Use pdflatex (or xelatex) instead.}
    \def\abort{\csname @@end\endcsname}
%   or \def\abort{\stop}
  \fi
\fi
\abort
...
\begin{document}
...
```

If desired, it would be simple to recast this as a package (a `.sty` file), which is what Lars exemplified in his response.

References

[1] Jean-François Burnol. Re: Arranging parallel texts. Post to `comp.text.tex` newsgroup, 24 February 2011.

[2] Stephen Hicks. Re: [texhax] multiple documents within a document. Post to `texhax` mailing list, 30 March 2010.

[3] Heiko Oberdiek. Re: How to make all images protrude into the outer border. Post to `comp.text.tex` newsgroup, 3 January 2010.

[4] Heiko Oberdiek. The ifpdf package, April 2012. `http://mirror.ctan.org/macros/latex/contrib/oberdiek`.

[5] Will Robertson. The `ifxetex` package, 2009. `http://mirror.ctan.org/macros/generic/ifxetex`.

[6] Thomas Thurman. Arranging parallel texts. Post to `comp.text.tex` newsgroup, 22 February 2011.

[7] Rasmus Villemoes. Aborting unless running pdflatex. Post to `comp.text.tex` newsgroup, 2 August 2010.

[8] Peter Wilson. The changepage package, 2009. `http://mirror.ctan.org/macros/latex/contrib/changepage/`.

[9] Peter Wilson. The memoir class for configurable typesetting, 2013. `http://mirror.ctan.org/macros/latex/contrib/memoir`.

[10] Peter Wilson. Parallel typesetting for critical editions: The ledpar package, 2014. `http://mirror.ctan.org/macros/latex/contrib/ledmac`.

◇ Peter Wilson
12 Sovereign Close
Kenilworth, CV8 1SQ
UK
herries dot press (at)
earthlink dot net

Online LaTeX editors and other resources

Paweł Łupkowski

Abstract

In this paper I will review several LaTeX editors and other resources available online. I will focus on the range of packages offered and compilation options available. The main question I aim to answer in this paper is how online LaTeX tools change the way we can work with LaTeX.

Introduction

The main question I would like to ask in this paper is how online LaTeX tools change the way we may work with LaTeX. To answer this question I will review several LaTeX editors and resources available online. I will focus mainly on writeLaTeX [8] and ShareLaTeX [5], which, in my opinion, are the most interesting online editors available.

Using LaTeX in online environment can have different motivations. First, it is very convenient for people working mainly with LaTeX. Online solutions offer this environment on virtually any machine with Internet access. Not only editing and compilation is possible, but also file storage in the cloud is available for these services. Also, online LaTeX editor makes collaboration easier (file sharing, version control, the same package set available for the collaborators).

Online LaTeX solutions can also be a great help in teaching LaTeX. They can be a fast and easy way to start editing and compiling documents without installation of the LaTeX environment and an editor — convenient for both teachers and students.

1 writeLaTeX and ShareLaTeX

WriteLaTeX and ShareLaTeX are mature but still rapidly developing projects. They are not only LaTeX editors but also offer a wide range of services in order to make working in an online environment easy and effective. Both environments offer:

- an editor (with syntax highlighting, line numbering and live preview),
- an online LaTeX compiler,
- file storage,
- document templates,
- sharing and collaboration options.

First let us take a look at the *Privacy and terms of service*, since we are offered an option of keeping our files in the cloud. For writeLaTeX, we read that:

> By using our Services you provide us with information, files, and folders that you submit to writeLaTeX (together, 'your stuff'). You retain full ownership to your stuff. We don't claim any ownership to any of it. [8]

ShareLaTeX providers also assure us that:

> You retain all ownership, copyright and intellectual property rights to any content uploaded to ShareLaTeX. Your content will only be shared with other users of your choosing and we will never share your content with third parties without your consent. [5]

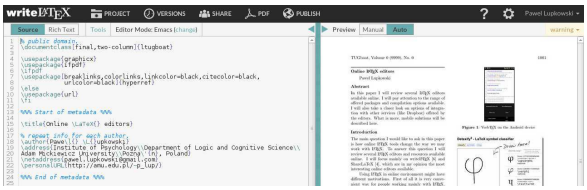


Figure 1: writeLaTeX interface

In what follows I will focus only on the functionality offered for writeLaTeX and ShareLaTeX's free accounts (they both offer various paid plans suited for different users' needs — more information can be found on the projects' websites).

The WriteLaTeX interface is presented in Figure 1. It has two panels — on the left is an editor and on the right a live preview. The compilation is done automatically; after each source change the preview is refreshed. You can hide either of the panels. More importantly, you can also turn off the live preview. One of the nice features of the writeLaTeX editor is that you can turn on emacs or vim mode to use the corresponding key-bindings. Unfortunately, spell checking is not available in the free account.

Recently a new feature has been added to the editor called *Rich Text*. This mode allows you to write your document in a semi-WYSIWYG fashion. A comparison of default mode and rich text mode is presented in Figure 2. This new mode is well designed and preserves a clear structure of a LaTeX code. In my opinion it might be useful for beginners, especially when editing complicated texts.

Another point I find interesting in this editor is that you do not have to register to use it. You can just start writing right away after visiting the web page. This opens a wide range of interesting applications. One such is to publish your document in writeLaTeX in such a way that it might be opened and edited by other users. This mechanism is used e.g. by the LaTeX template page described in Section 3.

As for ShareLaTeX, its interface is very similar to that of writeLaTeX (see Figure 3). At the start, it displays three panels — the leftmost one displays files used in a given project; the middle one is an editor

First published in BachoTeX 2014 proceedings, pp. 109–112. Reprinted with permission.

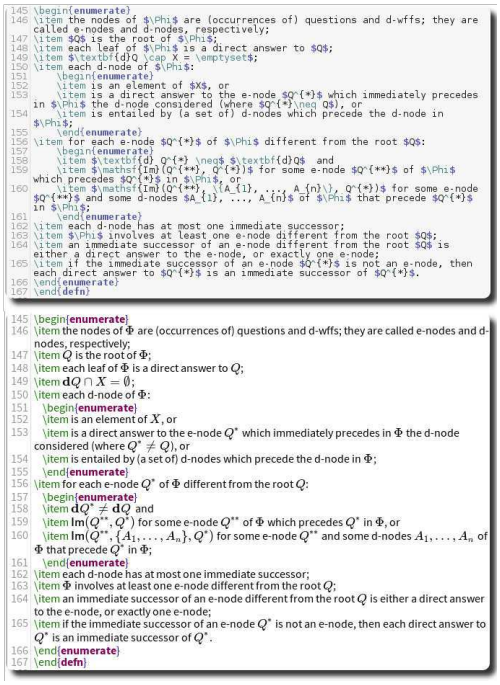


Figure 2: writeLaTeX interface — regular vs rich text mode example

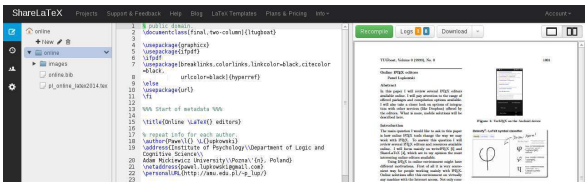


Figure 3: ShareLaTeX interface

and the rightmost is a preview. By default compilation is manual; you click "Recompile" to see changes in the preview. As in writeLaTeX you can turn on emacs or vim editor mode. It is worth stressing that registration is necessary to use ShareLaTeX.

Very usefully, ShareLaTeX offers different compilers to use for your documents (via the *Settings* section). Currently ShareLaTeX supports LaTeX, pdfLaTeX, XeLaTeX and LuaLaTeX. At present, writeLaTeX only supports pdfLaTeX, but adding XeLaTeX is planned (see [8, Help]).

Both writeLaTeX and ShareLaTeX are intuitive and easy to use and explore. Tutorials and help are also offered. A comparison of these editors in a variety of categories is presented in Table 1.

To conclude this section, it is worth mentioning another interesting project: LaTeXlab [2]. This editor is developed within Google Docs. One of its interesting features is that it allows using a local LaTeX installation as an compiler.

Table 1: writeLaTeX vs. ShareLaTeX (free accounts)

	writeLaTeX	ShareLaTeX
storage quota	100MB	no data
tag projects	yes	yes
Dropbox	no	no
compilers	pdfLaTeX	pdfLaTeX, LaTeX, XeLaTeX, LuaLaTeX
templates	yes	yes
`*.zip` upload	yes	yes
file history	yes	no
limited number of projects	no	no
mobile support	yes	poor
edit without registration	yes	no
collaboration	unlimited	with 1 user only

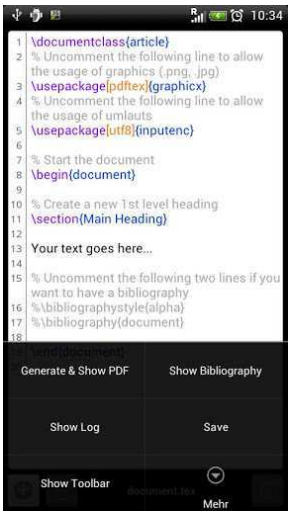


Figure 4: VerbTeX on the Android device

2 Go mobile

The editors presented in the previous section can be used on a mobile device (with a reasonably big screen). But there is a tool that is designed especially for mobile devices, namely VerbTeX [7]. VerbTeX is available for all popular mobile platforms, notably Android, iOS, and Windows 8.

VerbTeX works in two modes: local (storing your files on the mobile device) and cloud mode (keeping your files in the *verbosus.com* cloud). Files in local mode can be synchronised with a Dropbox account. Files are stored as projects. In the free version of this application you can have at most two documents in one project. VerbTeX allows you to create, store and edit your documents on the mobile

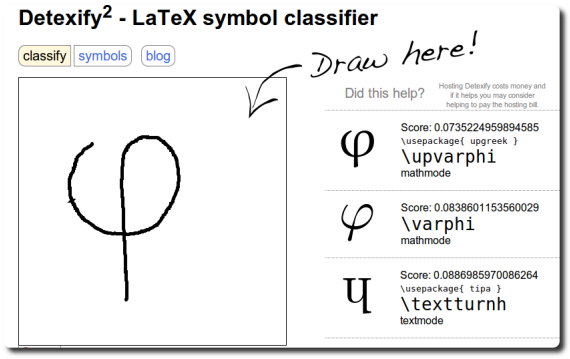


Figure 5: Detexify

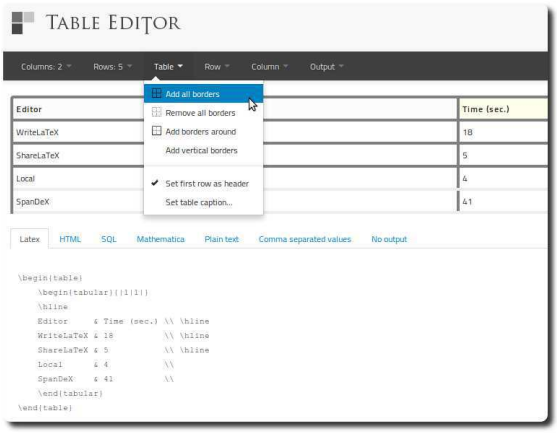


Figure 6: Online table editor

device. Previously existing files can be uploaded and to help in writing new documents, a custom *new.tex* document template is available. The editor offers syntax highlighting and line numbering, as can be noticed in Figure 4. Most important, to compile your document you will need an Internet connection. This keeps the application quite small — the Android version is only 1.7MB. The default compilation output is PDF.

3 Other online resources

Let's turn to other online resources useful for working with LaTeX. The tool I use the most is certainly *Detexify* [1]. The idea behind this tool is to make it easy to find a command for a given symbol listed in [4]. All you have to do is draw your desired symbol and then wait. You will obtain a list of recognised symbols with commands. Furthermore, each command is supplemented with some information about the package needed to use it. *Detexify* is presented in Figure 5.

Detexify enables a fast, efficient and very intuitive symbol search. The tool is also available for mobile devices running Android or iOS.

Another online resource worth mentioning here is the LaTeX templates website [3]. The page hosts a large (and still growing) number of LaTeX document templates grouped in useful categories, such as academic journals, articles, books, calendars, conference posters, etc. Each template is described in detail and supplemented with an example. A very nice feature is that the templates can be edited online using writeLaTeX by just a single click on the "Open with writeLaTeX" button, making the templates even more useful and easy to use.

Our list of useful LaTeX resources is closed by an online table editor. I often find it hard to easily design a large table. [6] can help in such a situation. This online tool helps you to design a table in a WYSIWYG fashion and then export it to LaTeX code (and other formats, such as HTML, plain text and Mathematica code). The tool allows you to decide on the table borders, merge cells, add caption and edit text align in cells. The interface is shown in Figure 6.

Of course the short list given here is far from being complete, but it represents resources that I find most useful in my everyday work.

Summary

The way we work today is heavily influenced by the Internet and more and more by mobile devices. Tools and resources described in this paper enable us to work with LaTeX according to these new trends. Document typesetting might be device-independent: we may easily create, edit and share document using a web browser or even a mobile phone. This opens new possibilities for LaTeX users, possibilities available already for users of WYSIWYG editors, like Office Online or Google Docs.

References

[1] Detexify. `detexify.kirelabs.org`.
[2] LaTeXlab. `docs.latexlab.org`.
[3] LaTeX templates website. `latextemplates.com`.
[4] Scott Pakin. The comprehensive LaTeX symbol list. `ctan.org/pkg/comprehensive`, 2009.
[5] ShareLaTeX. `www.sharelatex.com`.
[6] Table editing online. `truben.no/latex/table`.
[7] VerbTeX. `verbosus.com`.
[8] writeLaTeX. `www.writelatex.com`.

◇ Paweł Łupkowski
Institute of Psychology
Dept. of Logic and Cognitive Science
Adam Mickiewicz University
Poznań, Poland
`pawel dot lupkowski (at) gmail dot com`
`http://amu.edu.pl/~p_lup/`

Exporting XML and ePub from ConTEXt

Hans Hagen

1 Introduction

There is a pretty long tradition of typesetting math with TEX and it looks like this program will dominate for many more years. Even if we move to the web, the simple fact that support for MathML in some browsers is suboptimal will drive those who want a quality document to use PDF instead.

I'm writing this in 2014, at a time when XML is widespread. The idea of XML is that you code your data in a very structured way, so that it can be manipulated and (if needed) validated. Text has always been a target for XML which is a follow-up to SGML that was in use by publishers. Because HTML is less structured (and also quite tolerant with respect to end tags) we prefer to use XHTML but unfortunately support for that is less widespread.

Interestingly, documents are probably among the more complex targets of the XML format. The reason is that unless the author restricts him/herself or gets restricted by the publisher, tag abuse can happen. At Pragma we mostly deal with education-related XML and it's not always easy to come up with something that suits the specific needs of the educational concept behind a school method. Even if we start out nice and clean, eventually we end up with a polluted source, often with additional structure needed to satisfy the tools used for conversion.

We have been supporting XML from the day it showed up and most of our projects involve XML in one way or the other. That doesn't mean that we don't use TEX for coding documents. This manual is for instance a regular TEX document. In many ways a structured TEX document is much more convenient to edit, especially if one wants to add a personal touch and do some local page make-up. On the other hand, diverting from standard structure commands makes the document less suitable for output other than PDF. There is simply no final solution for coding a document, it's mostly a matter of taste.

So we have a dilemma: if we want to have multiple output, frozen PDF as well as less-controlled HTML output, we can best code in XML, but when we want to code comfortably we'd like to use TEX. There are other ways, like Markdown, that can be converted to intermediate formats like TEX, but that is only suitable for simple documents: the more advanced documents get, the more one has to escape from the boundaries of (any) document encoding, and then often TEX is not a bad choice. There is a good reason why TEX survived for so long.

It is for this reason that in ConTEXt MkIV we can export the content in a reasonable structured way to XML. Of course we assume a structured document. It started out as an experiment because it was relatively easy to implement, and it is now an integral component.

2 The output

The regular output is an XML file but as we have some more related data it gets organized in a tree. We also export a few variants. An example is given below:

```
./test-export
./test-export/images
./test-export/images/...
./test-export/styles
./test-export/styles/test-defaults.css
./test-export/styles/test-images.css
./test-export/styles/test-styles.css
./test-export/styles/test-templates.css
./test-export/test-raw.xml
./test-export/test-raw.lua
./test-export/test-tag.xhtml
./test-export/test-div.xhtml
```

Say that we have this input:

```
\setupbackend
  [export=yes]

\starttext
  \startsection[title=First]
    \startitemize
      \startitem one \stopitem
      \startitem two \stopitem
    \stopitemize
  \stopsection
\stoptext
```

The main export ends up in the `test-raw.xml` export file and looks like the following (we leave out the preamble and style references, and some line breaks are for *TUGboat*):

```
<document> <!-- with some attributes -->
  <section detail="section" chain="section"
           level="3">
    <sectionnumber>1</sectionnumber>
    <sectiontitle>First</sectiontitle>
    <sectioncontent>
      <itemgroup detail="itemize" chain="itemize"
                 symbol="1" level="1">
        <item>
          <itemtag><m:math ..><m:mo>‘</m:mo>
          </m:math></itemtag>
          <itemcontent>one</itemcontent>
        </item>
        <item>
          <itemtag><m:math ..><m:mo>‘</m:mo>
          </m:math></itemtag>
```

```
          <itemcontent>two</itemcontent>
        </item>
      </itemgroup>
    </sectioncontent>
  </section>
</document>
```

This file refers to the stylesheets and therefore renders quite well in a browser like Firefox that can handle XHTML with arbitrary tags.

The `detail` attribute tells us what instance of the element is used. Normally the `chain` attribute is the same but it can have more values. For instance, if we have:

```
\definefloat[graphic][graphics][figure]
...
\startplacefigure[title=First]
    \externalfigure[cow.pdf]
\stopplacefigure
...
\startplacegraphic[title=Second]
    \externalfigure[cow.pdf]
\stopplacegraphic
```

we get this:

```
<float detail="figure" chain="figure">
  <floatcontent>...</floatcontent>
  <floatcaption>...</floatcaption>
</float>
<float detail="graphic" chain="figure graphic">
  <floatcontent>...</floatcontent>
  <floatcaption>...</floatcaption>
</float>
```

This makes it possible to style specific categories of floats by using a (combination of) `detail` and/or `chain` as filters.

The body of the `test-tag.xhtml` file looks similar but it is slightly more tuned for viewing. For instance, hyperlinks are converted to a way that CSS and browsers like more. Keep in mind that the raw file can be the base for conversion to other formats, so that one stays closest to the original structure.

The `test-div.xhtml` file is even more tuned for viewing in browsers as it completely does away with specific tags. We explicitly don't map onto native HTML elements because that would make everything look messy and horrible, if only because there seldom is a relation between those elements and the original. One can always transform one of the export formats to pure HTML tags if needed.

```
<body>
  <div class="document">
    <div class="section" id="aut-1">
      <div class="sectionnumber">1</div>
      <div class="sectiontitle">First</div>
      <div class="sectioncontent">
        <div class="itemgroup itemize symbol-1">
          <div class="item">
            <div class="itemtag"><m:math ...>
              <m:mo>‘</m:mo></m:math></div>
            <div class="itemcontent">one</div>
          </div>
          <div class="item">
            <div class="itemtag"><m:math ...>
              <m:mo>‘</m:mo></m:math></div>
            <div class="itemcontent">two</div>
          </div>
        </div>
        <div class="float figure">
          <div class="floatcontent">...</div>
          </div>
          <div class="floatcaption">...></div>
        </div>
        <div class="float figure graphic">
          <div class="floatcontent">...</div>
          </div>
          <div class="floatcaption">...></div>
        </div>
      </div>
    </div>
  </body>
```

The default CSS file can deal with tags as well as classes. The file of additional styles contains definitions of so-called highlights. In the ConTEXt source one is better off using explicit named highlights instead of local font and color switches because these properties are then exported to the CSS. The images style defines all images used. The templates file lists all the elements used and can be used as a starting point for additional CSS styling.

Keep in mind that the export is *not* meant as a one-to-one visual representation. It represents structure so that it can be converted to whatever you like.

In order to get an export you must start your document with:

```
\setupbackend
  [export=yes]
```

So, we trigger a specific (extra) backend. In addition you can set up the export:

```
\setupexport
  [svgstyle=test-basic-style.tex,
   cssfile=test-extras.css,
   hyphen=yes,
   width=60em]
```

The `hyphen` option will also export hyphenation information so that the text can be nicely justified. The `svgstyle` option can be used to specify a file where math is set up; normally this would only contain a `bodyfont` setup, and this option is only needed if you want to create an ePub file afterwards which has math represented as SVG.

The value of `cssfile` ends up as a style reference in the exported files. You can also pass a comma-separated list of names (between curly braces). These entries come after those of the automatically generated CSS files so you need to be aware of default properties.

3 Images

Inclusion of images is done in an indirect way. Each image gets an entry in a special image related stylesheet and then gets referred to by `id`. Some extra information is written to a status file so that the script that creates ePub files can deal with the right conversion, for instance from PDF to SVG. Because we can refer to specific pages in a PDF file, this subsystem deals with that too. Images are expected to be in an `images` subdirectory and because in CSS the references are relative to the path where the stylesheet resides, we use `../images` instead. If you do some postprocessing on the files or relocate them you need to keep in mind that you might have to change these paths in the image-related CSS file.

4 ePub files

At the end of a run with exporting enabled you will get a message to the console that tells you how to generate an ePub file. For instance:

```
mtxrun --script epub --make --purge test
```

This will create a tree with the following organization:

```
./test-epub
./test-epub/META-INF
./test-epub/META-INF/container.xml
./test-epub/OEBPS
./test-epub/OEBPS/content.opf
./test-epub/OEBPS/toc.ncx
./test-epub/OEBPS/nav.xhtml
./test-epub/OEBPS/cover.xhtml
./test-epub/OEBPS/test-div.xhtml
./test-epub/OEBPS/images
./test-epub/OEBPS/images/...
./test-epub/styles
./test-epub/styles/test-defaults.css
./test-epub/styles/test-images.css
./test-epub/styles/test-styles.css
./test-epub/mimetype
```

Images will be moved to this tree as well and if needed they will be converted, for instance into SVG. Converted PDF files can have a `page-`⟨*number*⟩ in their name when a specific page has been used.

You can pass the option `--svgmath` in which case math will be converted to SVG. The main reason for this feature is that we found out that MathML support in browsers is not currently as widespread as might be expected. The best bet is Firefox which natively supports it. The Chrome browser had it for a while but it got dropped and math is now delegated to JavaScript and friends. In Internet Explorer MathML should work (but I need to test that again).

This conversion mechanism is kind of interesting: one enters TEX math, then gets MathML in the export, and that gets rendered by TEX again, but now as a standalone snippet that then gets converted to SVG and embedded in the result.

5 Styles

One can argue that we should use native HTML elements but since we don't have a good guaranteed-consistent mapping onto that, it makes no sense to do so. Instead, we rely on either explicit tags with details and chains or divisions with classes that combine the tag, detail and chain. The tagged variant has some more attributes and those that use a fixed set of values become classes in the division variant. Also, once we start going the (for instance) `H1`, `H2`, etc. route we're lost when we have more levels than that or use a different structure. If an `H3` can reflect several levels it makes no sense to use it. The same is true for other tags: if a list is not really a list than tagging it with `LI` is counterproductive. We're often dealing with very complex documents so basic HTML tagging becomes rather meaningless.

If you look at the division variant (this is used for ePub too) you will notice that there are no empty elements but `div` blocks with a comment as content. This is needed because otherwise they get ignored, which for instance makes table cells invisible.

The relation between `detail` and `chain` (reflected in `class`) can best be seen from the next example.

```
\definefloat[myfloata]
\definefloat[myfloatb][myfloatbs][figure]
\definefloat[myfloatc][myfloatcs][myfloatb]
```

This creates two new float instances. The first inherits from the main float settings, but can have its own properties. The second example inherits from the `figure` so in fact it is part of a chain. The third one has a longer chain.

```
<float detail="myfloata">...</float>
<float detail="myfloatb" chain="figure">
...</float>
<float detail="myfloatc" chain="figure myfloatb">
...</float>
```

In a CSS style you can now configure tags, details, and chains as well as classes (we show only a few possibilities). Here, the CSS element on the first line of each pair is invoked by the CSS selector on the second line.

```
div.float.myfloata { }
  float[detail='myfloata'] { }
div.float.myfloatb { }
  float[detail='myfloatb'] { }
div.float.figure { }
  float[detail='figure']   { }
div.float.figure.myfloatb { }
  float[chain~='figure'][detail='myfloata'] { }
div.myfloata { }
  *[detail='myfloata'] { }
div.myfloatb { }
  *[detail='myfloatb'] { }
div.figure { }
  *[chain~='figure'] { }
div.figure.myfloatb { }
  *[chain~='figure'][detail='myfloatb'] { }
```

The default styles cover some basics but if you're serious about the export or want to use ePub then it makes sense to overload some of it and/or provide additional styling. You can find plenty about CSS and its options on the Internet.

6 Coding

The default output reflects the structure present in the document. If that is not enough you can add your own structure, as in:

```
\startelement[question]
Is this right?
\stopelement
```

You can also pass attributes:

```
\startelement[question][level=difficult]
Is this right?
\stopelement
```

But these will be exported only when you also say:

```
\setupexport
  [properties=yes]
```

You can create a namespace. The following will generate attributes like `my-level`.

```
\setupexport
  [properties=my-]
```

In most cases it makes more sense to use highlights:

```
\definehighlight
  [important]
  [style=bold]
```

This has the advantage that the style and color are exported to a special CSS file.

Headers, footers, and other content that is part of the page builder are not exported. If your document has cover pages you might want to hide them too. The same is true when you create special chapter title rendering with a side effect that content ends up in the page stream. If something shows up that you don't want, you can wrap it in an `ignore` element:

```
\startelement[ignore]
Don't export this.
\stopelement
```

◇ Hans Hagen
Pragma ADE
http://pragma-ade.com
http://luatex.org

The box-glue-penalty algebra of TEX and its use of \prevdepth

Frank Mittelbach

Contents

Abstract

This article discusses certain aspects of TEX's approach to line breaking and its consequences for automatically calculating the right amount of vertical space between lines in more complex layouts.

It starts with giving a short introduction to the box-glue-penalty algebra used by TEX to model material to typeset. We then look at how the program calculates the vertical glue between lines in which the parameter \prevdepth plays a crucial role. Next we examine different types of output routines and evaluate how and to what extent the TEX algorithms can accommodate their goals.

The final conclusion is that this is an area where we can pose problems that cannot be resolved using current TEX, ε-TEX, pdfTEX, or XETEX, unless you restrict the allowable input, as there is no way to obtain some of the information used by TEX's algorithms for later manipulation of the result.

Like the answer to many questions these days, the situation is (probably) different with LuaTEX — probably, because I haven't actually tried it, but given the additional possibilities offered by LuaTEX a solution should be feasible.

1 The box-glue-penalty algebra

TEX's typesetting is built around a model that is known as the box-glue-penalty algebra [1]. At the lowest level we have (character) boxes that have (as far as TEX is concerned) no inner structure. These (character) boxes intermixed with glue (representing spaces) and penalties (representing possible break points) are what TEX combines to form higher level objects and eventually build up pages.

The first level of construction is called a horizontal list and such a list can either form a new box of its own (a horizontal box also known as an \hbox) or it can be passed to the paragraph builder that then (using a large number of parameters) will break the list apart into sub-lists (possibly dropping some content at the break points) to form the individual lines (again \hboxes) of a paragraph.

It is also possible to build vertically oriented lists, again consisting of boxes, glue and penalties. Here the glue represents vertical spaces and any penalties guide splitting the list later on. Such a list can become a box of its own (a vertical box or \vbox) or it can simply form a "galley" from which, by some method, TEX once in a while chops off a certain amount to form the content for a page to be produced.

Boxes in vertical lists are different though: while all lists can contain explicitly or implicitly constructed boxes, only horizontal lists can contain character boxes. If, while constructing a vertical list, TEX encounters a character box it puts the current construction on hold and starts building a horizontal list. If it then encounters a \par command (or an empty line) it will pass the constructed horizontal list to the paragraph builder. That in turn chops it up into individual lines and returns those as horizontal boxes (paragraph lines) intermixed with glue and penalties. These are then added into the vertical list and construction of the vertical list continues.

2 Splitting lists

Splitting of horizontal lists can only be done by passing the list to the paragraph builder. The result in that case is a vertical list (or rather something that will become part of a vertical list) and it contains a varying number of horizontal boxes forming the lines of the paragraph. In other words it is not possible directly to take a horizontal list (or box) and split it into two horizontal lists.

In the case of vertical lists the situation is slightly different: on the so-called "main vertical list" on which the material for pages is gathered, TEX monitors the amount of material being gathered and at certain points, either directed by some explicit penalty or simply because it decided that there is now enough material, it will chop off the right amount of material for a single page and then fire up a subroutine called the output routine to process that material and build a final page from it.

In addition to that, TEX offers the possibility to split off a chunk of a specified size from a given vertical box and place it into a new box. Technically speaking this is more or less what the output routine process on the main vertical list does when it gets fired up. The only difference is that on boxes this is an explicit command that needs to be invoked in the

programming code and it operates only on explicit boxes formed earlier.

It would be interesting to have the same functionality on the program level for horizontal boxes but for some reason that never made it into the program.

In other respects horizontal and vertical splitting is a very similar operation (on the box-glue-penalty algebra level). Splits can only happen at explicit penalties or at the left of glue provided it is immediately preceded by a box.[1] So in case of two globs of glue directly next to each other, a split can only happen before the first glue. Consecutive penalties behave like a single penalty unless they both force a break and we are in a horizontal list.[2]

Once a break point is chosen TEX drops all glue and penalties following it until it comes to the next box. The rationale behind this is that something like white space between words should vanish if you have a line break, and so should white space between lines if you have a page break.

This rather simple model allows to define surprisingly complex behavior, simply by specifying cleverly constructed sequences of glue, penalties and (empty) boxes. Appendix D in *The TEXbook* [2] shows a number of examples.

In summary, the box-glue-penalty model proved to be an ingenious way to model typesetting requirements and although it is not fully orthogonal and perhaps misses one or the other feature that would be useful, it gets the job done in a concise manner, and it is fair to say that even after more than three decades nobody has come up with anything better.

3 Assembling a vertical box or galley

A box in TEX terms is described by three dimensions: its height, depth, and width. The rationale is that characters that form words are lined up horizontally, each having a certain height but some of them stick out below the imaginary line (known as the baseline) and thus have a depth. Consequently, the boxes have a reference point at its left side with the material above the reference point forming the height of the box, and the material below, the depth.

If a horizontal list is used to form a new box then the inner boxes are lined up on their reference points, glue between the boxes appears as spaces, and the height and depth of the resulting box will be the maximum height and depth of the inner ones.[3] The width of the new box is simply the sum of all widths including the amounts taken up by the glue items. The reference point of the newly created box is then again on its left side at the baseline.

The situation with vertical lists is similar: they too align the boxes inside on their reference points (only this time vertically) and glue between boxes becomes spaces in the vertical direction. The width of the newly formed box is then the maximum width of its inner boxes. The calculation of height and depth, however, is slightly more complicated. By default, the depth will be the depth of the last box inside but only if this box is not followed by glue (a penalty would be allowed) — in the latter case the depth would be zero. The height is then calculated as the sum of all the heights and depths of all boxes plus the spaces and minus the box depth that was calculated earlier. In short you will end up with a box that has a very large height and a (normally) small depth.

In fact the depth of the new box is further restricted by a parameter called `\boxmaxdepth`: if the calculated depth exceeds this value then the reference point of the constructed box is lowered until the depth is no longer in violation. By default the value of this parameter is the largest possible dimension so that the restriction doesn't apply for manually created boxes unless this is changed.

On the main vertical list `\maxdepth` is used for the same purpose. If TEX decides that material needs to be packaged into a box to be passed to the output routine it uses that parameter to determine the maximum depth and thus the height of that box. In contrast to `\boxmaxdepth` this parameter typically has a setting that allows only for small depths to ensure that material is not "falling off the page" if it has an unusually large depth.

As an alternative to the construction explained above TEX also supports building a vertical box whose reference point aligns with the first box inside (`\vtop`), i.e., its height will be the height of this first box and the depth will hold everything else. If the list starts out with glue then the height of such a box will be zero.

[1] In real life the situation is, of course, more complex. TEX, for example, understands how to hyphenate words and so during the paragraph breaking it might introduce additional break points (and possibly even additional characters or variations into a list of character boxes), but abstractly one can think of this as just another version of boxes, glue and penalties that have been present from the beginning.

[2] The value of a penalty describes the desirability to break at a certain point (the smaller the better). The anomaly that two forcing penalties in horizontal mode behave differently and produce two breaks is due to some implementation decisions.

[3] As there are some operations to lower or raise boxes with respect to their reference point, this is not quite accurate, but one could consider such a manipulation simply as an operation that forms a new box with new dimensions.

4 Calculation of vertical glue

In the previous sections we explained how lists in the box-glue-penalty algebra are turned into new boxes. We also mentioned that the paragraph builder splits a horizontal list into a sequence of box-glue-penalty items but so far we haven't explained how this precisely happens and what kind of glue items are constructed during this process.

The main goal in paragraph building is to form lines of text that are (normally) vertically positioned in a way that the distance from baseline to baseline is constant. TEX manages this in the following way: whenever it builds a vertical list it keeps track of the depth of the last box appended to the list in a parameter called `\prevdepth`. At the beginning of a list it has a sentinel value of `-1000pt` to indicate that no box has been added so far.

When the paragraph builder builds a line box this box will have a certain height and depth. TEX then calculates the glue to be placed before the box by using the parameter that holds the standard baseline to baseline distance (`\baselineskip`) and subtracts from it the height of this box and the current value of `\prevdepth`. The resulting value is then appended as a glue item unless it is smaller than `\lineskiplimit` (i.e., the box height or the previous depth was very large). In that case TEX simply appends a glue item with a fixed value defined by another parameter called `\lineskip`.[4] The `\prevdepth` is then updated to hold the depth of the box just appended so that the calculation for the next line will be correct.

When a box is manually added to a vertical list, e.g., via `\box\mybox`, no baseline calculation happens and no glue gets prepended to the box. However, `\prevdepth` gets updated to hold the depth value of the newly appended box and thus any following box added by the paragraph builder would correctly calculate glue for baseline separation.

It is possible to inspect and even change the current value of `\prevdepth` within macro code while TEX is in vertical mode, i.e., while it is building a vertical list. Thus this only works between paragraphs and not within a paragraph as the paragraph builder acts on a horizontal list after all macro code has been expanded. In other words, you can determine the depth of the last line in a paragraph and by changing `\prevdepth`, pretend that it has a different value and thereby influence the baseline calculation for the first line of a following paragraph. However, you can't do the same for lines within the paragraph.

It is also very important to understand that this parameter is special in that it is local to the current vertical list being built and that it doesn't obey the normal scoping rules. So if you change it within a group it will keep its value when the group ends. Instead it only (and always!) reverts to a previous value if the construction of a vertical list has come to an end and TEX resumes building an outer vertical list, e.g., if boxes are nested within each other or if we return to the main vertical list after building a box or return from an output routine.

[4] Again this is a bit of an oversimplification. There are some more parameters involved and in certain circumstances nothing is appended. Also, if we are at the beginning of the list, or more precisely when `\prevdepth` has this special value, no glue is appended. For precise details see [2], especially chapters 12 and 14.

5 Standard output routines

TEX's paragraph builder is a sophisticated piece of software that uses dynamic programming to optimize the line breaking over the paragraph as a whole (with the sometimes surprising result that a change near the end modifies line breaking decisions much earlier on).

In contrast, TEX's page breaking concepts are much simpler. In essence, TEX assembles material on the main vertical list until it is clear that there is more material than can possibly fit on the page. At that moment TEX stops assembling material for the main vertical list and instead looks through all material gathered and decides on a final break point for the page using a number of parameters to guide this process. The material prior to this break point (which is a vertical list) is then packaged into a box (`\box255`) and a special piece of code, "the current output routine", is fired up.

The normal purpose of this output routine is to repackage and possibly embellish the material stored in `\box255`, e.g., by adding a running header or a page number, and then shipping it out to the output file. When everything has been done, control is given back to the process that fills the main vertical list and processing continues there.

As the lines of a paragraph are always added in one go to the main vertical list, TEX has typically accumulated more than it actually can use in the output routine. So when it returns from processing the page material, the main vertical list is not empty but contains a few boxes (and glue) that TEX had seen but decided not to use.

Furthermore, the output routine code is allowed to put some material from `\box255` back (typically after splitting it into several pieces) and in fact it can even generate new material to be put back into the main vertical list. To allow for this, TEX starts a new vertical list when the output routine starts and

the output routine can then place box-glue-penalty items into this list while working. Once the output routine has ended, this vertical list (if it contains anything) is placed at the head of the main vertical list, followed by any material already on it but not chosen for the current page.

Now what happens with `\prevdepth` during that time? When the output routine starts, it holds the depth of the last box contributed to the main vertical list, which may or may not be the last box that shows up in `\box255`. As the output routine starts a new vertical list, this value is shelved away and this new list gets its own instance starting out with `-1000pt` as usual. So if the output routine does something fairly complicated that includes building paragraphs, these paragraph lines are vertically spaced out using the rules explained above. Once the output routine ends, the value for `\prevdepth` from the main vertical list is restored.

This is normally the correct decision: if some material was not being used for the current page, then this will form the end of the main vertical list after the output routine has ended and thus `\prevdepth` will correctly reflect the depth of the last box appended there.

If on the other hand all material got used for the last page, then the value of `\prevdepth` no longer reflects the real situation as it still contains the depth from the last box. However, as long as the main vertical list is effectively empty at this point, this doesn't matter as TEX throws away any glue item after a page break until it sees the first box. It then inserts new glue, based on the height of this box and the current value of a parameter, namely `\topskip`. So all that happens is that the paragraph builder may have calculated a glue to go in front of the first paragraph line based on wrong assumptions but as this glue is thrown away later it doesn't matter.

6 Special output routines

But there is one further case: everything from the main vertical list got used but the output routine itself put something back. In that case the last box on the main vertical list will be whatever the output routine has deposited there, but the value of `\prevdepth` still reflects the last box that was there before the output routine was called.

The standard output routines in LATEX and plain TEX do not have this issue as they do not put anything back. However, the situation is quite different if you look at special output routines. These output routines typically get explicitly invoked by setting some explicit penalty and thus there will be no leftovers on the vertical list that correspond to the `\prevdepth` value.

For example, the multicol package, on reaching the end of a `multicols` environment, invokes an output routine that takes the gathered material, splits it up into balanced columns and then pushes the result back as a single block for reprocessing.

In that case the value of `\prevdepth` on the main vertical list will be the depth of the last box in the last column, but after balancing the overall depth of the result may very well be quite different (as the last column may be the shortest one, so its depth isn't even taken into account). As a result the baseline calculation of a following paragraph line will be based on wrong assumptions.

In fact multicol tried to account for this and added a negative space and then set `\prevdepth` to zero. However this happened inside the output routine so that the negative space survived but the value of `\prevdepth` got reverted after the output routine returned! (And it doesn't help to use `\global` from within the output routine as `\prevdepth` simply doesn't care.) As the difference is typically less than 2pt and multicol additionally adds a space of `\multicolsep` this bug remained undetected for a long time.

The solution to this problem then is, of course, to carefully keep track of what the output routine intends to put back, measure the final depth within the output routine and store it away in a global variable. Then, once the output routine has ended, explicitly set `\prevdepth` to the saved value to make it reflect the true situation.

7 An unsolvable problem?

The previous section explained how special output routines can be written to correctly reflect the situation with `\prevdepth`. But this requires that the output routine is always explicitly triggered — a situation in which we know that there is no remainder material that could throw us off track. But what happens if the output routine puts material back but is invoked asynchronously by the standard TEX mechanism?

For starters we then have a problem in regaining control after the output routine has ended though that can be resolved with a few tricks involving `\aftergroup` and a nested set of output routines.

But even if we do this we don't really know to what value we should set `\prevdepth`. It would need to be the depth of the material we put back in case there was nothing left on the main vertical list, but it needs to be left alone if this is not the case. And we can't arrange for the material returned to have

the same \prevdepth that was current before the output routine since we don't have access to this value within the output routine, and as the output routine is triggered asynchronously we can't (easily?) obtain it beforehand or as part of the process.

So this is something to ponder.

8 Some answers

Donald Knuth already gave a partial answer to this problem in *The TEXbook* [2, p. 262ff] where he discussed an output routine that adds index headings in random places in the text. The restriction in his algorithm is that the \prevdepth is assumed to be sufficiently small (that is, smaller than \maxdepth in fact). In that case we can use the depth of \box255 in the output routine as a measure for the \prevdepth calculations that will have taken place if there is any remainder in recent contributions, and use this to adjust the nominal depth of the material added to match that. And if there isn't any remainder then this doesn't really matter either.

However, this approach can't be used unmodified if this restriction isn't guaranteed to hold, i.e., if the material typeset may have arbitrary depth that is then masked by the page \maxdepth adjustment. Artificially enlarging \maxdepth is not an option either, as that would incorrectly alter the allowed page break positions.

A possible extension of the algorithm is to re-box the material inside the output routine to determine its natural depth, but unfortunately that turned out to be not enough to cover all cases.

So after a couple of false positives (i.e., pseudo-solutions that failed in one or another boundary case) my conclusion is that this problem cannot be solved within TEX or ε-TEX as long as the typesetting and line breaking is done by the engine. The main reason for this is the following:

- If TEX is doing the line breaking and automatically appends new material to the vertical lists it calculates the necessary glue based on the height of the newly appended box and the current \prevdepth value to achieve a baseline to baseline distance that corresponds to the value of \baselineskip.
- However, if that brings two boxes too close together it adds some extra glue (\lineskip).
- So if a "baseline skip" glue was added we can adjust its value based on the size of newly inserted material as we know the target size (i.e., \baselineskip) and the \prevdepth used (applying a variation of Knuth's algorithm outlined above).
- If, however, a \lineskip glue was added, our calculations are off base and there is no way within TEX to determine that we are in this branch of the typesetting algorithm short of disabling it and doing all box maneuvers manually.[5]

With LuaTEX the situation is different: With some moderate Lua programming effort it should be possible to traverse a node list, say the one stored in \splitdiscards, and determine if \lineskip was used. Depending on the scenario one could then either keep that node or delete it or replace it with an appropriate new value.

References

[1] Donald E. Knuth and Michael F. Plass. Breaking paragraphs into lines. In *Digital Typography*. CSLI Publications, Stanford, CA, USA, 1999.

[2] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading, MA, 1986.

◇ Frank Mittelbach
Mainz, Germany
http://www.latex-project.org

[5] I would be very much interested to be proven wrong here: If somebody finds a solution that covers the general case using just TEX or ε-TEX, please offer it as a *TUGboat* article.

The bird and the lion: arara

Paulo Roberto Massa Cereda

1 Prologue

There I was, back in 2011, with a huge project in my hands: a songbook. But it was far from any ordinary book due to the involved complexity: each song had several tags and at least 25 indexes, with different styles! Of course, TEX and friends were able to tackle this beast on their own, but I was not prepared. The lion was definitely hungry and I was the typographic meat provider.

My compilation workflow was striking: at least 30 to 40 steps in order to achieve the final result. As a first experiment, I wrote a nice `Makefile` and the problem had appeared to be solved once and for all. Suddenly, however, I found myself in need of a portable solution: I had to share my projects with at least three different operating systems (Windows, GNU/Linux and Mac OS X) and I should ensure that all the needed tools were in place for my workflow to work. Worse: I had to rely on system-dependent commands and other nuisances.

My first idea was to stand on the shoulders of giants and rely on the brilliant `latexmk` by John Collins; sadly, the workflow was too complicated for me to grasp at once, and my `.latexmkrc` shortly became a beast on its own. The second idea was to use `rubber` but, as my worst nightmares became true, at some point, I was writing ugly hacks and injecting Python code into the tool itself. Alas, no success, the songbook remained intractable.

When all else had failed, I decided to come up with a solution on my own. I sat in front of my computer with an open terminal and started to code while listening to Pink Floyd. In a couple of hours, a new tool was tackling my songbook.

I mentioned this journey in the chat room of the TEX community at StackExchange and Enrico Gregorio encouraged me to release this tool into the wild. Later on, Marco Daniel, Brent Longborough, Nicola Talbot and many, many others jumped in and a new project — arara — was born. The name was chosen as an homage to a Brazilian bird of the same name, which is a macaw. The word *arara* comes from the Tupian word *a'rara*, which means *big bird* (much to my chagrin, Sesame Street's iconic character Big Bird is not a macaw; according to some sources, he claims to be a golden condor). As I mention in the user manual, araras are colorful, noisy, naughty and very funny. Everybody loves araras. The name seemed catchy for a tool and, in the blink of an eye, arara was quickly spread to the whole TEX world. It is an interesting story of a bird and a lion living together.

2 The basics

I think the best way to explain how arara works is to provide a quick comparison with similar tools, like the ones I've mentioned in the prologue. Let us use the following file `hello.tex` as an example:

```
\documentclass{article}
\begin{document}
Hello world!
\end{document}
```

How would one successfully compile `hello.tex` with `latexmk` and `rubber`, for instance? It's quite straightforward: it is just a matter of providing the file to the tool and letting it do the hard work; a simple `latexmk hello` or `rubber -pdf hello` would do the trick. Now, if one tries `arara hello`, I'm afraid *nothing* will be generated; the truth is, arara doesn't know what to do with your file (and the tool will raise an error message complaining about this issue). You need to tell arara what to do.

That is the major difference of arara when compared to other tools: it is not an automatic process and the tool does not employ any guesswork on its own. You are in control of your documents; arara won't do anything unless you teach it how to do a task and explicitly tell it to execute the task.

How does one teach arara how to do a task? The answer is quite simple: we have to define rules. A rule is a formal description of how arara should handle a certain task. For example, if we want to use `pdflatex` with arara, we need a rule for that. Once a rule is defined, arara automatically provides an access layer to the user. The package provides dozens of predefined rules, so you already have several options out of the box to set up your workflow.

Once we know how to execute a task, we need to explicitly tell arara when to do it. This is done through a directive. A directive is a special comment inserted in the source file in which you indicate how arara should behave. You can insert as many directives as you want, and in any position of the file; arara will read the whole file and extract the directives. A directive should be placed in a line of its own, in the form

`% arara:` ⟨*directive*⟩

It is important to observe that a directive is not the command to be executed, but the name of the

rule associated with that directive (once arara finds a directive, it will look for the associated rule). That is basically how arara works: we teach the tool to do a task by providing a rule, and tell it to execute it via directives in the source code.

Sometimes, we need to provide additional information to the rule from the source code. That's why arara offers two types of directives:

empty directive An empty directive, as the name indicates, has only the rule identifier. The syntax for an empty directive is

`% arara:` ⟨*directive*⟩

parameterized directive A parameterized directive has the rule identifier followed by its arguments. It's very important to mention that the arguments are mapped by their identifiers and not by their positions. The syntax for such a directive is

`% arara:` ⟨*directive*⟩`: { `⟨*arglist*⟩` }`

An individual argument has the form

⟨*key*⟩`:` ⟨*value*⟩

and an ⟨*arglist*⟩ has keys with their respective values separated by commas. The arguments are defined according to the rule mapped by the directive (you cannot give an argument `foo` to a directive `bar` if it does not offer support for this named parameter).

If you want to disable a directive, there's no need to remove it from the source file. Simply replace

```
% arara:
```

by, for example,

```
% !arara:
```

or insert some other symbol before `arara:` and this directive will be ignored. The tool always looks for a line that, after removing the leading and trailing spaces, starts with a comment and contains '`arara:`' as a word of its own. The user manual shows how to override this search pattern, but the `arara:` keyword is always required.

Now that we know how to tell arara what to do with `hello.tex`, we need to modify it a little by including the proper `pdflatex` directive:

```
% arara: pdflatex
\documentclass{article}
\begin{document}
Hello world!
\end{document}
```

And that's it. Now, calling `arara hello` (or `arara hello.tex` — both will work), the document will be successfully compiled. Then, let's say we would like to enable shell escape for this particular compilation; we can achieve that by providing a parameterized directive, like this:

```
% arara: pdflatex: { shell: yes }
\documentclass{article}
\begin{document}
Hello world!
\end{document}
```

Of course, `shell` is defined in the rule scope, otherwise arara would raise an error about an invalid key. The user manual has a list of all available keys for each predefined rule.

As we've noted, arara relies on the provided source file as the main document. The `pdflatex` rule above thus passes the provided filename to the `pdflatex` command. Let us see how to override such information in order to run programs on other files.

There's a reserved argument key named `files`, whose value is a list. If you want to override the default value of the main document for a specific rule, use this key in the directive, in the form

`% arara:` ⟨*directive*⟩`: { files: [`⟨*list*⟩`] }`

For example, if you need to run `makeindex` on files `a` and `b` instead of the default `hello`, you can use

```
% arara: makeindex: { files: [ a, b ] }
```

That is the trick I used when working with 25 indexes in my songbook: it was just a matter of providing their names and which styles to the `makeindex` directive.

There is much more to arara than what I've described in this section. For more complete coverage of available tools, please refer to the user manual. arara is already available in TeX Live and also as a standalone tool. Source code is available at

`https://github.com/cereda/arara`

It is also important to observe that a new version is in the works and this hopefully will fix a couple of nuisances found with the current official version (namely, version 3.0 of the tool). The new version also includes several improvements which will be unveiled as soon as the tool reaches its official release (as a bonus, a new article will be provided for readers).

3 Examples

Now that we know how arara works, let us see some examples. The first document, `ex1.tex`, requires two runs in order to set the labels correctly, so we write two directives.

```
% arara: pdflatex
% arara: pdflatex
```

```
\documentclass{article}
\begin{document}
\section{Introduction}
\label{sec:intro}
As seen in Section~\ref{sec:intro}\ldots
\end{document}
```

The second document, `ex2.tex`, has a citation (courtesy of `xampl.bib`, available in the TEX Live tree), so we need to specify a call to `bibtex` as well:

```
% arara: pdflatex
% arara: bibtex
% arara: pdflatex
% arara: pdflatex
\documentclass{article}
\begin{document}
As seen in \cite{book-full}\ldots
\bibliographystyle{plain}
\bibliography{xampl}
\end{document}
```

The third document, `ex3.tex`, has the same LATEX source as the previous example, but we want to use `biber` instead of `bibtex`; it's just a matter of replacing the directive:

```
% arara: pdflatex
% arara: biber
% arara: pdflatex
% arara: pdflatex
\documentclass{article}
\usepackage{biblatex}
\addbibresource{xampl.bib}
\begin{document}
As seen in \cite{book-full}\ldots
\printbibliography
\end{document}
```

The fourth document, `ex4.tex`, shows an example of a simple index, so we include a `makeindex` directive:

```
% arara: pdflatex
% arara: makeindex
% arara: pdflatex
\documentclass{article}
\usepackage{makeidx}
\makeindex
\begin{document}
Some text.\index{Apple}
\printindex
\end{document}
```

The fifth document, `ex5.tex`, shows a glossary, courtesy of the great `glossaries` package. We need to add a `makeglossaries` directive for this:

```
% arara: pdflatex
% arara: makeglossaries
% arara: pdflatex
\documentclass{article}
\usepackage{glossaries}
\newglossaryentry{equation}{name=equation,
description={an equation usually involves
at least one variable, and has two sides;
typically we will try to solve an
equation for one of the unknown
variables}}
\makeglossaries
\begin{document}
\glsaddall
\printglossary
\end{document}
```

The sixth document, `ex6.tex`, shows a good old plain TEX source, compiled with the `tex` directive. As expected, we will get `ex6.dvi` as output.

```
% arara: tex
Hello world.
\bye
```

The seventh document, `ex7.tex`, enhances the previous example by adding a conversion chain in order to obtain a PDF file; this is done by converting `ex7.dvi` to `ex7.ps` and then to `ex7.pdf` (the directive names are self-explanatory).

```
% arara: tex
% arara: dvips
% arara: ps2pdf
Hello world.
\bye
```

The eighth document, `ex8.tex`, uses a package (namely `minted`) which requires shell escapes to be enabled. We give the (parameterized) directive for that in order to achieve a proper compilation:

```
% arara: pdflatex: { shell: yes }
\documentclass{article}
\usepackage{minted}
\begin{document}
\begin{minted}{c}
int main() {
  printf("hello, world");
  return 0;
}
\end{minted}
\end{document}
```

The ninth document, `ex9.tex`, uses `multibib` in order to provide two separate bibliographies; we must run `bibtex` on the second auxiliary file `A.aux` as well, so we give the special `files` key to `bibtex`:

```
% arara: pdflatex
% arara: bibtex
% arara: bibtex: { files: [ A ] }
```

```
% arara: pdflatex
% arara: pdflatex
\documentclass{article}
\usepackage{multibib}
\newcites{A}{References 2}

\begin{document}
\cite{book-full}
\citeA{inproceedings-full}

\bibliographystyle{plain}
\begingroup
\bibliography{xampl}
\endgroup

\bibliographystyleA{plain}
\begingroup
\bibliographyA{xampl}
\endgroup
\end{document}
```

Observe the `\begingroup` and `\endgroup` around the `\bibliography` commands: this is because the sample bibliography file `xampl.bib` has a preamble field in which a couple of commands are defined which would otherwise cause some ugly definition errors (as both `.bbl` files contain `\newcommand`).

Alternatively, we could have used one `bibtex` directive with two `files`:

```
% arara: bibtex: { files: [ ex9, A ] }
```

instead of writing two `bibtex` directives. However, I would choose to write a separate line for each `bibtex` run, both to better organize my workflow, and to provide only the second auxiliary filename; otherwise, the main document filename would also have to be explicitly specified.

The tenth and last document, `ex10.tex`, has a `clean` directive to remove `ex10.log` after correctly generating the PDF file:

```
% arara: pdftex
% arara: clean: { files: [ ex10.log ] }
Hello world.
\bye
```

And that is it: arara is quite straightforward to use, provided that you know the available rules and keys, and also the compilation workflow needed.

4 Final remarks

As shown in this article, arara can be used in complex workflows, such as theses and books. You can tell the tool to compile a document, generate indexes and apply styles, remove temporary files, compile other documents, run METAFONT or METAPOST, create glossaries, call `pdfcrop`, `gnuplot`, move files, and much more. Furthermore, arara is platform-independent. It's all up to you.

That said, I believe that the warning featured in the user manual still applies: Hic Sunt Dracones. Hopefully the new version will exterminate a couple of nuisances and bugs found in the current official release; however, as with any non-trivial software, the tool is far from being bug-free. And you will learn that arara gives you plenty of rope. In other words, you will be responsible for how the tool behaves and all the consequences from your actions. Sorry to sound scary, but I really needed to tell you this. After all, one of arara's greatest features is the freedom it offers. But as you know, freedom always comes at a cost.

Feedback is surely welcome for me to improve this humble tool — just write an e-mail to me or any other member of the team and we will reply as soon as possible. The source code is fully available at

`https://github.com/cereda/arara`

Feel free to contribute to the project by forking it, submitting bugs, sending pull requests or even translating it to your language. If you want to support the LaTeX development with a donation, the best way to do this is by donating to the TeX Users Group.

Happy TeXing with arara!

◇ Paulo Roberto Massa Cereda
Analândia, São Paulo, Brazil
`cereda (at) users dot sf dot net`

The SWIGLIB project*

Luigi Scarso

Abstract

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

1 Introduction

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

Analogous to the `\usepackage` macro of LATEX, it is easy to extend the built-in features of Lua by means of external Lua modules, usually loaded with `load("⟨module_name⟩")`. What perhaps is less well known is that the same is also available for *binary* modules; for example, a C library compiled in the native format of the platform. This is due to the double nature of Lua, as both an interpreted language and a library that can be linked with an application (see [4, p. 249]):[1] the interaction of the Lua library and the application must follow the application programming interface (API) of Lua.

While for LuaTEX there is currently no official C API — it's a program, not a library — the Lua API is completely described in the Lua manual (`http://www.lua.org/manual`) and it counts 245 items, including constants, macros, functions and standard libraries. They consistently use a stack to exchange data (and hence several functions are dedicated to the stack manipulation) and use an opaque data structure to store the current state, but the stack is accessible only by the state and sometimes it is confused with it. By design (related to the choice of ANSI C for the implementation language) the Lua state is not thread-safe, but the library is carefully designed to avoid destructive interference in global variables and in some cases multithreading with a single shared state appears to be possible [13]. In any case, the best solution is to avoid sharing the state between multiple threads — the library can in fact safely manage different states, at the price of more complex code.

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

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

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

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

Figure 1: Calling a Lua function from C.

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

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

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

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

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

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

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

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

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

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

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

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

Figure 2: Calling a C function from Lua.

2 The SWIG tool

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

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

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

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

The next subsections will try to address these questions.

2.1 Generate a wrapper

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

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

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

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

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

the SWIG interface file to create the wrapper is:

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

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

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

and, supposing that the application header and the shared library `myapplication.so` live in the current

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

directory `./`, the binary module `core.so` is compiled as below (again for the Linux platform):

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

and loaded in Lua with

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

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

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

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

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

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

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

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

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

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

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

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

Second, by adding C functions and data structures to the inline interface a user can build a custom *usermodule*, eventually using other application libraries. In other words, SWIG also supports interface files `usercore.i`, `usernative.i`, `userinline.i` and

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

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

[5] Although the chapter "Variable Length Arguments" at `http://www.swig.org/Doc3.0/SWIGDocumentation.html#Varargs` does start with *a.k.a. "The horror. The horror."*

`userluacode.i` and hence a `usercore` binary module that stays in the same directory as the `core` application.

2.2 Compilation of a wrapper

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

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

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

and for Windows 64-bit it's almost the same:

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

A simple bash script should be easily ported to different platforms: the GNU Autotools are well suited for Unix-like systems, but Windows has its own toolchain and such a a shell script can be translated in a batch script without much effort, giving a good starting point (see [1, p. 3]).

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

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

Given that an application module always needs to resolve the Lua API symbols, the first consequence is that the `luatex` Windows binary must be compiled with a dynamic link to an external Lua library (a Lua DLL) and the same DLL must be used at build-time for the application module. Under Linux the Lua

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

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

```
#include <gmpxx.h>
#include <iostream>
using namespace std;
int main(void) {
    size_t a = 5;
    mpz_class b(a);
    cout << b.get_ui() << endl;
    cout<< sizeof a <<endl;
    return 0;
}
```

3 Deployment

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

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

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

```
curl
└─ 7.40.0
    ├─ docs
    ├─ linux
    │   ├─ resources
    │   │   ├─ include32
    │   │   ├─ include64
    │   │   │   └─ curl
    │   │   ├─ lib32
    │   │   └─ lib64
    │   └─ test
    │   build-linux-x86_64.sh
    ├─ osx
    │   └─ resources
    │       ├─ include32
    │       ├─ include64
    │       ├─ lib32
    │       └─ lib64
    │
    └─ windows
        ├─ resources
        │   ├─ include32
        │   │   └─ curl
        │   ├─ include64
        │   │   └─ curl
        │   ├─ lib32
        │   └─ lib64
        └─ test
        build-mingw32.sh
        build-mingw64.sh
        build-msys32.sh
        build-msys64.sh
    core.i
    inline.i
```

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

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

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

```
luacode.i
native.i
```

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

```
└─ luatex-beta-0.79.3.1
    ├─ include
    │   ├─ lauxlib.h
    │   ├─ luaconf.h
    │   ├─ lua.h
    │   ├─ lua.hpp
    │   ├─ lualib.h
    │   ├─ patch-01-utf-8
    │   ├─ patch-02-FreeBSD
    │   └─ patch-03-export
    ├─ linux
    │   └─ luatex
    ├─ w32
    │   ├─ libkpathsea-6.dll
    │   ├─ luatex.exe
    │   └─ texlua52.dll
    └─ w64
        ├─ libkpathsea-6.dll
        ├─ luatex.exe
        └─ texlua52.dll
```

3.1 Application module location in the TDS

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

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

SWIGLIB doesn't require a particular method to load a wrapper module, because this is a task of the format. The tests in the Subversion repository use the low-level Lua function `load`, but they need to know the system and the full path of the module; on the other hand, ConTEXt has a global `swiglib` function (see `util-lib.lua` and [3]) that is independent from the system and the path — but it doesn't use the `kpse` library.

4 Conclusions

Without a doubt, building a wrapper module requires a working knowledge at least of the C language, for which [5] is still a pleasure to read; useful information on shared libraries is also in [2] and [7] while for Linux [6] is still one of the best references, as [10] and [9] are for Windows. Moreover, having a working wrapper is only half of the story: the rest is a working Lua/TEX layer that suits with the format in use — and this cannot be part of the underlying SWIGLIB. The example with GMP 6.0.0 shows that an application module that compiles well and passes all the tests can still fail to compile an apparently innocuous program. The C code itself is not always easy to understand, as for example with the following program

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

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

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

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

LuaTEX as a general-purpose scripting language, i.e. to manage the installation of TEX packages.

Regarding LuaJITTEX [12]: even when it's possible to use the same interface file, the API and the `luajitex` libraries are not the same. Furthermore, LuaJIT users seem to prefer the use of the LuaJIT `ffi` module, which is roughly similar to SWIG. It should still be doable to implement in SWIG via a new backend LuaJIT-ffi that emits `ffi` chunks instead of the LuaJIT C API, effectively eliminating the need for a C compiler. Clearly work for the future.

Some practical examples of applications are shown in [3] and [11]. These will also be the subject of a future paper.

References

[1] John Calcote. *Autotools: A Practitioner's Guide to GNU Autoconf, Automake, and Libtool*. No Starch Press, San Francisco, CA, USA, 1st edition, 2010.

[2] Ulrich Drepper. How to write shared libraries. `http://www.akkadia.org/drepper/dsohowto.pdf`, December 10 2011. Accessed: 2015-03-5.

[3] Hans Hagen. Swiglib basics. `http://minimals.contextgarden.net/current/doc/context/pragma/general/manuals/swiglib-mkiv.pdf`.

[4] Roberto Ierusalimschy. *Programming in Lua, Third Edition*. Lua.Org, 3rd edition, 2013.

[5] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd edition, 1988.

[6] Michael Kerrisk. *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press, San Francisco, CA, USA, 1st edition, 2010.

[7] John R. Levine. *Linkers and Loaders*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1999.

[8] T. Przechlewski, editor. *What Can Typography Gain from Electronic Media?* Polska Grupa Użytkowników Systemu TEX — GUST, 2014. ISBN 9788393901609. `http://books.google.it/books?id=abDNoAEACAAJ`.

[9] Mark E. Russinovich, David A. Solomon, and Alex Ionescu. *Windows Internals, Part 1: Covering Windows Server 2008 R2 and Windows 7*. Microsoft Press, 6th edition, 2012.

[10] Mark E. Russinovich, David A. Solomon, and Alex Ionescu. *Windows Internals, Part 2: Covering Windows Server 2008 R2 and Windows 7 (Windows Internals)*. Microsoft Press, 2012.

[11] Luigi Scarso. Extending ConTEXt MkIV with PARI/GP. *ArsTEXnica*, 11:65–74, April 2011. `http://www.guitex.org/home/images/ArsTeXnica/AT011/AT11-scarso.pdf`.

[12] Luigi Scarso. LuaJITTEX. *TUGboat*, 34(1):64–71, 2013. `http://tug.org/TUGboat/34-1/tb106scarso.pdf`.

[13] Luigi Scarso. Some experiments with OpenMP and LuaTEX. In Przechlewski [8]. `http://www.gust.org.pl/bachotex/2014-pl/presentations/openmp-slides.pdf`.

◇ Luigi Scarso
`luigi dot scarso (at) gmail dot com`
`http://swiglib.foundry.supelec.fr`

Still tokens: LuaTEX scanners

Hans Hagen

1 Introduction

Tokens are the building blocks of the input for TEX and they drive the process of expansion which in turn results in typesetting. If you want to manipulate the input, intercepting tokens is one approach. Other solutions are preprocessing or writing macros that do something with their picked-up arguments. In ConTEXt MkIV we often forget about manipulating the input but manipulate the intermediate typesetting results instead. The advantage is that only at that moment do you know what you're truly dealing with, but a disadvantage is that parsing the so-called node lists is not always efficient and it can even be rather complex, for instance in math. It remains a fact that until LuaTEX version 0.80 ConTEXt hardly used the token interface.

In version 0.80 a new scanner interface was introduced, demonstrated by Taco Hoekwater at the ConTEXt conference 2014. Luigi Scarso and I integrated that code and I added a few more functions. Eventually the team will kick out the old token library and overhaul the input-related code in LuaTEX, because no callback is needed any more (and also because the current code still has traces of multiple Lua instances). This will happen stepwise to give users who use the old mechanism an opportunity to adapt.

Here I will show a bit of the new token scanners and explain how they can be used in ConTEXt. Some of the additional scanners written on top of the built-in ones will probably end up in the generic LuaTEX code that ships with ConTEXt.

2 The TEX scanner

The new token scanner library of LuaTEX provides a way to hook Lua into TEX in a rather natural way. I have to admit that I never had any real demand for such a feature but now that we have it, it is worth exploring.

The TEX scanner roughly provides the following sub-scanners that are used to implement primitives: keyword, token, token list, dimension, glue and integer. Deep down there are specific variants for scanning, for instance, font dimensions and special numbers.

A token is a unit of input, and one or more characters are turned into a token. How a character is interpreted is determined by its current catcode. For instance a backslash is normally tagged as 'escape character' which means that it starts a control sequence: a macro name or primitive. This means that once it is scanned a macro name travels as one token through the system. Take this:

```
\def\foo#1{\scratchcounter=123#1\relax}
```

Here TEX scans `\def` and turns it into a token. This particular token triggers a specific branch in the scanner. First a name is scanned with optionally an argument specification. Then the body is scanned and the macro is stored in memory. Because '`\scratchcounter`', '`\relax`', and '`#1`' are turned into tokens, this body has 7 tokens.

When the macro `\foo` is referenced the body gets expanded which here means that the scanner will scan for an argument first and uses that in the replacement. So, the scanner switches between different states. Sometimes tokens are just collected and stored, in other cases they get expanded immediately into some action.

3 Scanning from Lua

The basic building blocks of the scanner are available at the Lua end, for instance:

```
\directlua{print(token.scan_int())} 123
```

This will print `123` to the console. Or, you can store the number and use it later:

```
\directlua{SavedNumber = token.scan_int())} 123
We saved: \directlua{tex.print(SavedNumber)}
```

The number of scanner functions is (on purpose) limited but you can use them to write additional ones as you can just grab tokens, interpret them and act accordingly.

The `scan_int` function picks up a number. This can also be a counter, a named (math) character or a numeric expression. In TEX, numbers are integers; floating-point is not supported naturally. With `scan_dimen` a dimension is grabbed, where a dimen is either a number (float) followed by a unit, a dimen register or a dimen expression (internally, all become integers). Of course internal quantities are also okay. There are two optional arguments, the first indicating that we accept a filler as unit, while the second indicates that math units are expected. When an integer or dimension is scanned, tokens are expanded till the input is a valid number or dimension. The `scan_glue` function takes one optional argument: a boolean indicating if the units are math.

The `scan_toks` function picks up a (normally) brace-delimited sequence of tokens and (LuaTEX 0.80) returns them as a table of tokens. The function `get_token` returns one (unexpanded) token while `scan_token` returns an expanded one.

Because strings are natural to Lua we also have `scan_string`. This one converts a following brace-delimited sequence of tokens into a proper string.

The function `scan_keyword` looks for the given keyword and when found skips over it and returns `true`. Here is an example of usage:[1]

```
function ScanPair()
  local one = 0
  local two = ""
  while true do
    if token.scan_keyword("one") then
      one = token.scan_int()
    elseif token.scan_keyword("two") then
      two = token.scan_string()
    else
      break
    end
  end
  tex.print("one: ",one,"\\par")
  tex.print("two: ",two,"\\par")
end
```

This can be used as:

```
\directlua{ScanPair()}
```

You can scan for an explicit character (class) with `scan_code`. This function takes a positive number as argument and returns a character or `nil`.

1	0	escape
2	1	begingroup
4	2	endgroup
8	3	mathshift
16	4	alignment
32	5	endofline
64	6	parameter
128	7	superscript
256	8	subscript
512	9	ignore
1024	10	space
2048	11	letter
4096	12	other
8192	13	active
16384	14	comment
32768	15	invalid

So, if you want to grab the character you can say:

```
local c = token.scan_code(2^10 + 2^11 + 2^12)
```

In ConTEXt you can say:

```
local c = tokens.scanners.code(
  tokens.bits.space +
  tokens.bits.letter +
  tokens.bits.other
)
```

When no argument is given, the next character with catcode letter or other is returned (if found).

[1] In LuaTEX 0.80 you should use `newtoken` instead of `token`.

In ConTEXt we use the `tokens` namespace which has additional scanners available. That way we can remain compatible. I can add more scanners when needed, although it is not expected that users will use this mechanism directly.

(new)token	tokens.	arguments
	scanners.boolean	
scan_code	scanners.code	(bits)
scan_dimen	scanners.dimension	(fill,math)
scan_glue	scanners.glue	(math)
scan_int	scanners.integer	
scan_keyword	scanners.keyword	
	scanners.number	
scan_token	scanners.token	
scan_tokens	scanners.tokens	
scan_string	scanners.string	
	scanners.word	
get_token	getters.token	
set_macro	setters.macro	(catcodes,cs, str,global)

All except `get_token` (or its alias `getters.token`) expand tokens in order to satisfy the demands.

Here are some examples of how we can use the scanners. When we would call `Foo` with regular arguments we do this:

```
\def\foo#1{%
  \directlua {
    Foo("whatever","#1",{n = 1})
  }
}
```

but when `Foo` uses the scanners it becomes:

```
\def\foo#1{%
  \directlua{Foo()} {whatever} {#1} n {1}\relax
}
```

In the first case we have a function `Foo` like this:

```
function Foo(what,str,n)
  -- do something with these three parameters
end
```

and in the second variant we have (using the `tokens` namespace):

```
function Foo()
  local what = tokens.scanners.string()
  local str  = tokens.scanners.string()
  local n    = tokens.scanners.keyword("n") and
               tokens.scanners.integer() or 0
  -- do something with these three parameters
end
```

The string scanned is a bit special as the result depends on what is seen. Given the following definition:

```
          \def\bar  {bar}
\unexpanded\def\ubar {ubar}
% that's \protected in e-tex etc.
          \def\foo  {foo-\bar-\ubar}
```

```
\def\wrap {{foo-\bar}}
\def\uwrap{{foo-\ubar}}
```

We get:

`foo`	foo
`foo-\bar`	foo-bar
`foo-\ubar`	foo-\ubar
`foo-\bar`	foo-bar
`foo-\ubar`	foo-ubar
`foo$bar$`	foo*bar*
`\foo`	foo-bar-ubar
`\wrap`	foo-bar
`\uwrap`	foo-\ubar

Because scanners look ahead the following happens: when an open brace is seen (or any character marked as left brace) the scanner picks up tokens and expands them unless they are protected; so, effectively, it scans as if the body of an `\edef` is scanned. However, when the next token is a control sequence it will be expanded first to see if there is a left brace, so there we get the full expansion. In practice this is convenient behaviour because the braced variant permits us to pick up meanings honouring protection. Of course this is all a side effect of how TEX scans.[2]

With the braced variant one can of course use primitives like `\detokenize` and `\unexpanded` (in ConTEXt: `\normalunexpanded`, as we already had this mechanism before it was added to the engine).

4 Considerations

Performance-wise there is not much difference between these methods. With some effort you can make the second approach faster than the first but in practice you will not notice much gain. So, the main motivation for using the scanner is that it provides a more TEX-ified interface. When playing with the initial version of the scanners I did some tests with performance-sensitive ConTEXt calls and the difference was measurable (positive) but deciding if and when to use the scanner approach was not easy. Sometimes embedded Lua code looks better, and sometimes TEX code. Eventually we will end up with a mix. Here are some considerations:

- In both cases there is the overhead of a Lua call.
- In the pure Lua case the whole argument is tokenized by TEX and then converted to a string that gets compiled by Lua and executed.
- When the scan happens in Lua there are extra calls to functions but scanning still happens in TEX; some token to string conversion is avoided and compilation can be more efficient.
- When data comes from external files, parsing with Lua is in most cases more efficient than parsing by TEX.
- A macro package like ConTEXt wraps functionality in macros and is controlled by key/value specifications. There is often no benefit in terms of performance when delegating to the mentioned scanners.

Another consideration is that when using macros, parameters are often passed between `{}`:

```
\def\foo#1#2#3%
  {...}
\foo {a}{123}{b}
```

and suddenly changing that to

```
\def\foo{\directlua{Foo()}}
```

and using that as:

```
\foo {a} {b} n 123
```

means that `123` will fail. So, eventually you will end up with something:

```
\def\myfakeprimitive{\directlua{Foo()}}
\def\foo#1#2#3{\myfakeprimitive {#1} {#2} n #3 }
```

and:

```
\foo {a} {b} {123}
```

So in the end you don't gain much here apart from the fact that the fake primitive can be made more clever and accept optional arguments. But such new features are often hidden for the user who uses higher-level wrappers.

When you code in pure TEX and want to grab a number directly you need to test for the braced case; when you use the Lua scanner method you still need to test for braces. The scanners are consistent with the way TEX works. Of course you can write helpers that do some checking for braces in Lua, so there are no real limitations, but it adds some overhead (and maybe also confusion).

One way to speed up the call is to use the `\luafunction` primitive in combinations with predefined functions and although both mechanisms can benefit from this, the scanner approach gets more out of that as this method cannot be used with regular function calls that get arguments. In (rather low level) Lua it looks like this:

```
luafunctions[1] = function()
```

[2] This lookahead expansion can sometimes give unexpected side effects because often TEX pushes back a token when a condition is not met. For instance when it scans a number, scanning stops when no digits are seen but the scanner has to look at the next (expanded) token in order to come to that conclusion. In the process it will, for instance, expand conditionals. This means that intermediate catcode changes will not be effective (or applied) to already-seen tokens that were pushed back into the input. This also happens with, for instance, `\futurelet`.

```
  local a token.scan_string()
  local n token.scan_int()
  local b token.scan_string()
  -- whatever --
end
```

And in TEX:

```
\luafunction1 {a} 123 {b}
```

This can of course be wrapped as:

```
\def\myprimitive{\luafunction1 }
```

5 Applications

The question now pops up: where can this be used? Can you really make new primitives? The answer is yes. You can write code that exclusively stays on the Lua side but you can also do some magic and then print back something to TEX. Here we use the basic token interface, not ConTEXt:

```
\directlua {
local token = newtoken or token
function ColoredRule()
  local w, h, d, c, t
  while true do
    if token.scan_keyword("width") then
      w = token.scan_dimen()
    elseif token.scan_keyword("height") then
      h = token.scan_dimen()
    elseif token.scan_keyword("depth") then
      d = token.scan_dimen()
    elseif token.scan_keyword("color") then
      c = token.scan_string()
    elseif token.scan_keyword("type") then
      t = token.scan_string()
    else
      break
    end
  end
  if c then
    tex.sprint("\\color[",c,"]{"); end
  if t == "vertical" then
    tex.sprint("\\vrule")
  else
    tex.sprint("\\hrule")
  end
  if w then
    tex.sprint("width ",w,"sp"); end
  if h then
    tex.sprint("height ",h,"sp"); end
  if d then
    tex.sprint("depth ",d,"sp"); end
  if c then
    tex.sprint("\\relax}"); end
end
}
```

This can be given a TEX interface like:

```
\def\myhrule{\directlua{ColoredRule()}
             type {horizontal} }
\def\myvrule{\directlua{ColoredRule()}
             type {vertical} }
```

And then used as:

```
\myhrule width \hsize height 1cm color {darkred}
```

giving (grayscaled for *TUGboat* on paper, sorry):

Of course ConTEXt users can use the following commands to color an otherwise-black rule (likewise):

```
\blackrule[width=\hsize,height=1cm,
           color=darkgreen]
```

The official ConTEXt way to define such a new command is the following. The conversion back to verbose dimensions is needed because we pass back to TEX.

```
\startluacode
local myrule = tokens.compile {
  {
    { "width",  "dimension", "todimen" },
    { "height", "dimension", "todimen" },
    { "depth",  "dimension", "todimen" },
    { "color",  "string" },
    { "type",   "string" },
  }
}

interfaces.scanners.ColoredRule = function()
  local t = myrule()
  context.blackrule {
    color  = t.color,
    width  = t.width,
    height = t.height,
    depth  = t.depth,
  }
end
\stopluacode
```

With:

```
\unprotect \let\myrule\scan_ColoredRule \protect
```

and

```
\myrule width \textwidth height 1cm
        color {darkblue} \relax
```

we get:

There are many ways to use the scanners and each has its charm. We will look at some alternatives from the perspective of performance. The timings are more meant as relative measures than absolute

ones. After all it depends on the hardware. We assume the following shortcuts:

```
local scannumber  = tokens.scanners.number
local scankeyword = tokens.scanners.keyword
local scanword    = tokens.scanners.word
```

We will scan for four different keys and values. The number is scanned using a helper `scannumber` that scans for a number that is acceptable for Lua. Thus, `1.23` is valid, as are `0x1234` and `12.12E4`.

```
function getmatrix()
  local sx, sy = 1, 1
  local rx, ry = 0, 0
  while true do
    if scankeyword("sx") then
      sx = scannumber()
    elseif scankeyword("sy") then
      sy = scannumber()
    elseif scankeyword("rx") then
      rx = scannumber()
    elseif scankeyword("ry") then
      ry = scannumber()
    else
      break
    end
  end
  -- action --
end
```

Scanning the following specification 100000 times takes 1.00 seconds:

```
sx 1.23 sy 4.5 rx 1.23 ry 4.5
```

The "tight" case (no spaces) takes 0.94 seconds:

```
sx1.23 sy4.5 rx1.23 ry4.5
```

We can compare this to scanning without keywords. In that case there have to be exactly four arguments. These have to be given in the right order which is no big deal as often such helpers are encapsulated in a user-friendly macro.

```
function getmatrix()
  local sx, sy = scannumber(), scannumber()
  local rx, ry = scannumber(), scannumber()
  -- action --
end
```

As expected, this is more efficient than the previous examples. It takes 0.80 seconds to scan this 100000 times:

```
1.23 4.5 1.23 4.5
```

A third alternative is the following:

```
function getmatrix()
  local sx, sy = 1, 1
  local rx, ry = 0, 0
  while true do
    local kw = scanword()
    if kw == "sx" then
      sx = scannumber()
    elseif kw == "sy" then
      sy = scannumber()
    elseif kw == "rx" then
      rx = scannumber()
    elseif kw == "ry" then
      ry = scannumber()
    else
      break
    end
  end
  -- action --
end
```

Here we scan for a keyword and assign a number to the right variable. This one call happens to be less efficient than calling `scan_keyword` 10 times $(4 + 3 + 2 + 1)$ for the explicit scan. This run takes 1.11 seconds for the next line. The spaces are really needed as words can be anything that has no space.[3]

```
sx 1.23 sy 4.5 rx 1.23 ry 4.5
```

Of course these numbers need to be compared to a baseline of no scanning (i.e. the overhead of a Lua call which here amounts to 0.10 seconds. This brings us to the following table.

keyword checks	0.9 sec
no keywords	0.7 sec
word checks	1.0 sec

The differences are not that impressive given the number of calls. Even in a complex document the overhead of scanning can be negligible compared to the actions involved in typesetting the document. In fact, there will always be some kind of scanning for such macros so we're talking about even less impact. So you can just use the method you like most. In practice, the extra overhead of using keywords in combination with explicit checks (the first case) is rather convenient.

If you don't want to have many tests you can do something like this:

```
local keys = {
  sx = scannumber,
  sy = scannumber,
  rx = scannumber,
  ry = scannumber,
}

function getmatrix()
  local values = { }
  while true do
    for key, scan in next, keys do
      if scankeyword(key) then
        values[key] = scan()
```

[3] Hard-coding the word scan in a C code helper makes little sense, as different macro packages can have different assumptions about what a word is. And we don't extend LuaTEX for specific macro packages.

```
      else
        break
      end
    end
  end
  -- action --
end
```

This is still quite fast although one now has to access the values in a table. Working with specifications like this is clean anyway so in ConTEXt we have a way to abstract the previous definition.

```
local specification = tokens.compile {
  {
    { "sx", "number" },
    { "sy", "number" },
    { "rx", "number" },
    { "ry", "number" },
  },
}

function getmatrix()
  local values = specification()
  -- action using values.sx etc --
end
```

Although one can make complex definitions this way, the question remains if it is a better approach than passing Lua tables. The standard ConTEXt way for controlling features is:

```
\getmatrix[sx=1.2,sy=3.4]
```

So it doesn't matter much if deep down we see:

```
\def\getmatrix[#1]{%
 \getparameters[@@matrix][sx=1,sy=1,
                          rx=1,ry=1,#1]%
   \domatrix
     \@@matrixsx
     \@@matrixsy
     \@@matrixrx
     \@@matrixry
   \relax}
```

or:

```
\def\getmatrix[#1]{%
 \getparameters[@@matrix][sx=1,sy=1,
                          rx=1,ry=1,#1]%
   \domatrix
     sx \@@matrixsx
     sy \@@matrixsy
     rx \@@matrixrx
     ry \@@matrixry
   \relax}
```

In the second variant (with keywords) can be a scanner like we defined before:

```
\def\domatrix#1#2#3#4%
  {\directlua{getmatrix()}}
```

but also:

```
\def\domatrix#1#2#3#4%
  {\directlua{getmatrix(#1,#2,#3,#4)}}
```

given:

```
function getmatrix(sx,sy,rx,ry)
    -- action using sx etc --
end
```

or maybe nicer:

```
\def\domatrix#1#2#3#4%
  {\directlua{domatrix{
     sx = #1, sy = #2,
     rx = #3, ry = #4
   }}}
```

assuming:

```
function getmatrix(values)
    -- action using values.sx etc --
end
```

If you go for speed the scanner variant without keywords is the most efficient one. For readability the scanner variant with keywords or the last shown example where a table is passed is better. For flexibility the table variant is best as it makes no assumptions about the scanner — the token scanner can quit on unknown keys, unless that is intercepted of course. But as mentioned before, even the advantage of the fast one should not be overestimated. When you trace usage it can be that the (in this case matrix) macro is called only a few thousand times and that doesn't really add up. Of course many different sped-up calls can make a difference but then one really needs to optimize consistently the whole code base and that can conflict with readability. The token library presents us with a nice chicken–egg problem but nevertheless is fun to play with.

6 Assigning meanings

The token library also provides a way to create tokens and access properties but that interface can change with upcoming versions when the old library is replaced by the new one and the input handling is cleaned up. One experimental function is worth mentioning:

```
token.set_macro("foo","the meaning of bar")
```

This will turn the given string into tokens that get assigned to `\foo`. Here are some alternative calls:

```
set_macro("foo")
≡ \def \foo {}
set_macro("foo", "meaning")
≡ \def \foo {meaning}
set_macro("foo", "meaning", "global")
≡ \gdef \foo {meaning}
```

The conversion to tokens happens under the current catcode regime. You can enforce a different regime by passing a number of an allocated catcode

table as the first argument, as with `tex.print`. As we mentioned performance before, setting at the Lua end like this:

```
token.set_macro("foo","meaning")
```

is about two times as fast as:

```
tex.sprint("\\def\\foo{meaning}")
```

or (with slightly more overhead) in ConTEXt terms:

```
context("\\def\\foo{meaning}")
```

The next variant is actually slower (even when we alias `setvalue`):

```
context.setvalue("foo","meaning")
```

but although 0.4 versus 0.8 seconds looks like a lot on a TEX run I need a million calls to see such a difference, and a million macro definitions during a run is a lot. The different assignments involved in, for instance, 3000 entries in a bibliography (with an average of 5 assignments per entry) can hardly be measured as we're talking about milliseconds. So again, it's mostly a matter of convenience when using this function, not a necessity.

7 Conclusion

For sure we will see usage of the new scanner code in ConTEXt, but to what extent remains to be seen. The performance gain is not impressive enough to justify many changes to the code but as the low-level interfacing can sometimes become a bit cleaner it will be used in specific places, even if we sacrifice some speed (which then probably will be compensated for by a little gain elsewhere).

The scanners will probably never be used by users directly simply because there are no such low level interfaces in ConTEXt and because manipulating input is easier in Lua. Even deep down in the internals of ConTEXt we will use wrappers and additional helpers around the scanner code. Of course there is the fun-factor and playing with these scanners is fun indeed. The macro setters have as their main benefit that using them can be nicer in the Lua source, and of course setting a macro this way is also conceptually cleaner (just like we can set registers).

Of course there are some challenges left, like determining if we are scanning input of already converted tokens (for instance in a macro body or token list expansion). Once we can properly feed back tokens we can also look ahead like `\futurelet` does. But for that to happen we will first clean up the LuaTEX input scanner code and error handler.

◇ Hans Hagen
Pragma ADE
http://pragma-ade.com
http://luatex.org

The Treasure Chest

Special note for this installment: please see Barbara Beeton's editorial column for some CTAN news and action recommendations.

The following is a list of selected new packages posted to CTAN (`http://ctan.org`) from October 2014 through March 2015, with descriptions based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. A few entries which the editors subjectively believe to be of especially wide interest or otherwise notable are starred; of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the TEX community. Comments are welcome, as always.

◇ Karl Berry

biblio

`bibfilex` in `biblio`
GUI bibliography manager written in Free Pascal.

fonts

`fontmfizz` in `fonts`
Access MFizz font icons in LaTeX.

* `newtxsf` in `fonts`
Sans serif math fonts based on `newtxmath` and STIX.

graphics

`ticollege` in `graphics/pgf/contrib`
Draw scientific calculator keys in Ti*k*Z.

`tikz-dimline` in `graphics/pgf/contrib`
Draw technical dimension lines in Ti*k*Z.

`tikz-palattice` in `graphics/pgf/contrib`
Draw particle accelerator lattices in Ti*k*Z.

`tipfr` in `graphics/pgf/contrib`
Output menu items, screenshots, and calculator keys in Ti*k*Z.

info

`latexcheat-de` in `info/latexcheat`
German adaptation of the English LaTeX cheat sheet.

macros

`musixtnt` in `macros`
MusiXTEX extension library enabling transformations of the effect of notes commands.

macros/generic

`apnum` in `macros/generic`
Arbitrary-precision numbers in pure TEX.

macros/latex

** `latex/base`
`latex/doc`
`latex/required/cyrillic`
`latex/required/graphics`
`latex/required/tools` in `macros/latex`
A major update to LaTeX 2_ε, by default incorporating changes previously included only by explicitly loading the `fixltx2e` package. A new `latexrelease` package and other mechanisms allow for controlling this. The included *LaTeX News* #22 has details, and additional articles are expected for the next *TUGboat*. This LaTeX release will be included in TEX Live 2015 and its pretests, and not distributed (in TEX Live) before that.

The `psnfss` and `babel` packages, though also required parts of LaTeX, are maintained separately from base LaTeX, and thus are not changed in this update (and they still work, too).

The announcement for CTAN: `lists.dante.de/pipermail/ctan-ann/2015-March/008366.html`.

macros/latex/contrib

`avremu` in `macros/latex/contrib`
Microprocessor simulation in pure LaTeX.

`bankstatement` in `macros/latex/contrib`
Generate bank statements from CSV data.

`basicarith` in `macros/latex/contrib`
Typeset textbook-style basic arithmetic.

`begingreek` in `macros/latex/contrib`
Typeset Greek in `pdflatex`.

`bondgraphs` in `macros/latex/contrib`
Draw bond graphs using Ti*k*Z.

`bookcover` in `macros/latex/contrib`
Typeset book covers and dust jackets.

`boxedminipage2e` in `macros/latex/contrib`
Framed minipages of a specified total width.

`calculation` in `macros/latex/contrib`
Typeset reasoned calculations (calculational proofs).

`cryptocode` in `macros/latex/contrib`
Typeset pseudocode, protocols, game-based proofs and black-box reductions in cryptography.

`cyber` in `macros/latex/contrib`
Annotate compliance with cybersecurity requirements.

`cybercic` in `macros/latex/contrib`
"Controls in Contents" for the `cyber` package.

`datetime2` in `macros/latex/contrib`
Formatting dates, times, etc.

`datetime2-*` in `macros/latex/contrib`
Language modules for `datetime2`.

`doclicense` in `macros/latex/contrib`
Putting documents under Creative Commons licenses.

ebproof in macros/latex/contrib
Typeset formal proofs in the style of sequent calculus.
ekaia in macros/latex/contrib
Format for the Basque scientific journal *Ekaia*.
elzcards in macros/latex/contrib
Typeset business cards, index cards, flash cards.
etdipa in macros/latex/contrib
Lightweight template for scientific documents.
europasscv in macros/latex/contrib
Support for the 2013 Europass CV standard.
fancyslides in macros/latex/contrib
Custom presentation class built on beamer.
*fcolumn in macros/latex/contrib
New column type f for typesetting financial tables from raw data.
fei in macros/latex/contrib
Support for the FEI University Center (Brazil) style.
fixocgx in macros/latex/contrib
Support ocgx in all known engines.
gender in macros/latex/contrib
Promote gender neutrality in gendered languages.
genealogytree in macros/latex/contrib
Pedigree and genealogical tree diagrams.
glossaries-* in macros/latex/contrib
Language modules for glossaries.
gsemthesis in macros/latex/contrib
Geneva School of Economics and Management PhD thesis format.
havannah in macros/latex/contrib
Board diagrams for the games Havannah and Hex.
indextools in macros/latex/contrib
Fixed imakeidx with bidi support.
jslectureplanner in macros/latex/contrib
Generate and manage university course material.
jumplines in macros/latex/contrib
Newspaper-style teasers with later continuations.
leadsheets in macros/latex/contrib
Typeset leadsheets and songbooks.
ndsu-thesis in macros/latex/contrib
North Dakota State University disquisition class.
prftree in macros/latex/contrib
Typeset natural deduction proofs.
romanbarpagenumber in macros/latex/contrib
Typesetting roman page numbers with bars.
sduthesis in macros/latex/contrib
Thesis template for Shandong University.
sesamanuel in macros/latex/contrib
Support for Sesamath Society books and papers.
turabian-formatting in macros/latex/contrib
Chicago-style formatting based on Turabian's work.
urcls in macros/latex/contrib
Support for University of Regensburg styles.
versonotes in macros/latex/contrib
Display brief notes on verso pages.
*xcolor-solarized in macros/latex/contrib
Defines the 16 colors from Schoonover's solarized palette.
xprintlen in macros/latex/contrib
Print TEX lengths in a variety of units.

macros/latex/contrib/babel-contrib

babel-bosnian in m/l/c/babel-contrib
Babel support for Bosnian.

macros/latex/contrib/beamer-contrib

epyt in m/l/c/beamer-contrib
Simple and clean theme for beamer.

macros/latex/contrib/biblatex-contrib

citeall in m/l/c/biblatex-contrib
Cite all entries of a bbl file created with BIBLATEX.

macros/plain

epsf-dvipdfmx in macros/plain/contrib
Supplement for epsf.tex when using dvipdfmx with non-origin EPS images.

macros/xetex

interchar in macros/xetex/latex
Managing character class schemes in XETEX.
*xespotcolor in macros/xetex/latex
Spot colors in XELATEX.

support

crossrefware in support
Scripts for working with crossref.org.
ctan_chk in support
gawk script for verification of CTAN uploads.
epspdf-setup in support
Standalone epspdf Windows executable.
hook-pre-commit-pkg in support
Pre-commit git hook to check LATEX syntax.
lug in support
Shell script to update TEX Local User Group web pages from the LUG database.

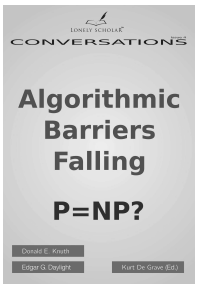
web

yacco2 in web
LR(1) compiler-compiler that emits literate grammars.

Book review: *Algorithmic Barriers Falling: P=NP?*

David Walden

Donald E. Knuth and Edgar G. Daylight, *Algorithmic Barriers Falling: P=NP?* Lonely Scholar, 2014, 116 pp. Paperback, US$20.00. ISBN 978-94-9138-604-8.



This is the second booklet-length interview of Donald Knuth by Edgar Daylight (done in June 2014). (For a review of the prior booklet, see `http://tug.org/TUGboat/tb34-3/tb108reviews-knuth.pdf`.)

Daylight is on a mission to further his "understanding of computer science by analyzing and documenting its past" (`http://tug.org/interviews/daylight.html`). To this end he has interviewed several pioneers of computer science and published the interviews (`http://www.walden-family.com/ieee/daylight-knuth.pdf`). Also to this end, Daylight and his editor Kurt De Grave have established a small publishing company for Daylight's work.

This second interview of Knuth moves back and forth between discussion of the early days of computer science and Knuth's current feelings about topics such as the writing of computing history and whether P = NP.

As with the first Knuth–Daylight interview booklet, this interview is interesting, easy to read, and relevant to the world of TEX.

In chapter 1, Knuth mentions how in the 1960s he decided to call the work he liked to do "analysis of algorithms" and hints that *Analysis of Algorithms* would have been a more appropriate name for his series of books titled *The Art of Computer Programming*.

In chapter 2, Knuth discusses his views on the writing of computing history (and the writing of science history more generally). He uses this discussion to include something he left out of his 2014 Stanford lecture titled "Let's Not Dumb Down the History of Computer Science" (`https://www.youtube.com/watch?v=gAXdDEQveKw`), a presentation that caused a lot of debate within the `sigcis.org` discussion group of historians of computing. Knuth returns to this topic again in chapter 6. (Chapter 2 also touches on the development of TEX.)

Chapter 3, 4, 5, and 7 discuss various topics in the early history of computer science.

Chapter 8 is about the development of TEX and literate programming. Some parts of this are already familiar to those of us who have read about Knuth's creation of TEX, but it also emphasizes how he moved from his original idea of trying "to express letters mathematically by measuring photographic images" (that didn't work out well) to the idea of "capturing the intelligence of design instead of the outcome of the design." He also explains the double meaning of the word "strokes" in the dedication of *The METAFONTbook*: "to Herman Zapf, whose strokes are the best"; Zapf not only draws beautiful strokes — he also stroked Knuth in the form of positive and negative critique. With regard to literate programming, I didn't previously know that Knuth was partially influenced in the METAFONT creative effort by a report by P.-A. de Marneffe titled *Holon programming: A survey*.

Chapter 9 discusses the problem of whether or not P = NP and Knuth's current opinion that P does equal NP. This chapter finishes with Knuth noting that he will write no more published papers (only books). He says his last paper was the one published in *TUGboat* that was the transcript of his talk on iTEX (ding) at the TEX Users Group 2010 annual conference in San Francisco (`http://tug.org/TUGboat/tb31-2/tb98knut.pdf`). He sees that 2010 humorous paper as the proper bookend for his first paper, also humorous, published 50 years ago in *Mad Magazine*.

The booklet also has a 116-element list of references and a nice index.

I recommend this interview booklet to several different classes of readers:

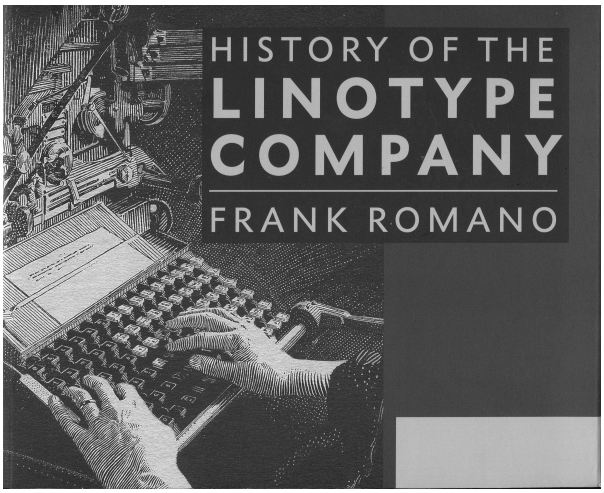
- Computing historians and students of computing history who want to read another first hand account touching on the early days of computer science, or who are interested in what an eminent computer scientist says about writing computing history.
- Historians and computing people who are considering taking advantage of the relative ease with which one can today publish a monograph with worldwide distribution (e.g., Amazon.com) without involving the academic presses.
- And, of course, those of us who are interested in all things Knuthian.

◇ David Walden
`http://www.walden-family.com/texland`

Book review: *History of the Linotype Company*

Boris Veytsman

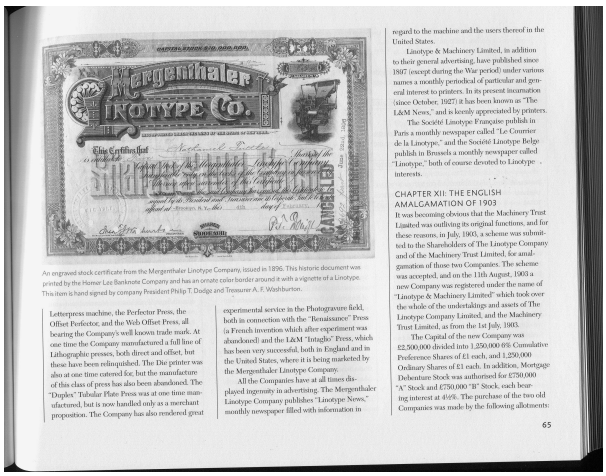
Frank Romano, *History of the Linotype Company*. RIT Press, 2014. 480 pp. Softcover, US$39.99. ISBN 978-1-933360-60-7.



Few inventions changed the cultural and political landscape as profoundly as those involving cheap and quick copying of words. Gutenberg's movable type made books available to multitudes of people with enormous consequences for civilization. However, manual composition was still too slow and expensive to mass-produce cheap newspapers. The invention of hot-metal typesetting by Ottmar Mergenthaler and his contemporaries made printing much faster and cheaper. It is difficult to fully quantify the consequences since it coincided with the advent of radio, and both these events had a similar effect: the deep penetration of news into everyday life. However, there is little doubt that newspapers and cheap books were very important in the history of the last century. For about one hundred years — until the advent of digital typesetting — Linotype machines ruled the world of mass-produced copy. While newspapers were the first adopters of the new technology, many others followed; as Frank Romano writes in his book, during World War II every US warship larger than cruiser class had a Linotype machine on board.

The Mergenthaler Linotype Company, established in 1886 (as Mergenthaler Printing Company) has a rich and interesting history. It included tumultuous arguments with the eponymous inventor, lawsuits, patent fights, mergers, acquisitions — and also technological innovations, hard workers, great artists and daring visionaries. Besides development and promotion of hot-metal typesetting, the company created an enormous number of typefaces, pioneered teletypesetting (again of huge importance for the newspaper business), experimented with phototypesetting and contributed to digital composition. The experience of its engineers with precision mechanical devices allowed the company to venture into other areas, including the production of bombsights and other armaments (especially during the wars).

Frank Romano, now an Emeritus Professor with RIT, worked at Linotype for eight years. He has written a brilliant book about the company — not a dry list of milestones, but rather a work of love and appreciation. The book includes, in excerpts or in full, rare or previously unpublished documents, such as the autobiography of Ottmar Mergenthaler, manuscripts, letters, earning reports, court filings, newspaper articles, brochures, author's own interviews, and many others. The book is lavishly illustrated, with hundreds of reproductions of samples, advertisements, photographs, books and other materials that Linotype published over 127 years of its existence.

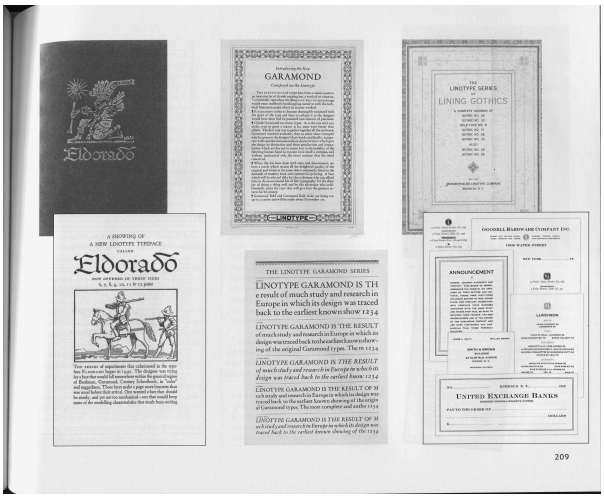


Romano's foremost interest is in the people at Linotype. The book has many vivid biographical sketches of extraordinary individuals who worked for the company or otherwise influenced it: from Ottmar Mergenthaler himself to the reclusive investor Gurdon Wattles (by the way, a role model for Warren Buffett). The people described in the book do not appear as cartoonish figures on a backdrop of Linotype's history: Romano has a rare ability to portray all his personages, even those mentioned only briefly, as alive and real. The human side of the history is his strongest feature.

Another interest for the author is evidently the financial side of company activity: he writes about its mergers, acquisitions and loans with attention to detail. If one wants to know the net income of the Mergenthaler Company in 1967 or how many

founders' shares were issued at the company incorporation, the information can be found in the book. The author also describes the lawsuits Linotype was involved in, including those about font copyright, which are still very relevant in the US.

The book devotes many pages to the influence of the company on the art of typography. Romano reproduces cover pages and spreads of the famous Linotype manuals, which defined the trends for contemporary typesetting. He describes the huge work the company did in the design of typefaces. Among the people of Linotype described in the book are Chauncey H. Griffin, Harry L. Cage, Paul A. Bennett, Mike Parker, and many other figures in the world of font design and typesetting.



Having worked at Linotype from 1959 to 1967, from mail boy to assistant ad manager, Romano devotes a chapter of his book to his personal reminiscences. It describes the life at the great company in the Sixties with loving detail: from the quality of food in the nearby restaurants to the generosity of expense accounts to the typical day in the ad department. This chapter is great reading for any lover of history.

Romano also briefly touches on the history of technology: hot-metal typesetting and other relevant inventions, such as punch-cutting machines and double-wedge spaceband. I wish, however, he had been as detailed here as in other parts of his book. I feel the book could have been improved by the inclusion of historical drawings, e.g., from the company's patent applications.

The book has many useful appendices, including a detailed index, lists of typefaces originated or used at Linotype (both alphabetical and chronological), a time table and a large bibliography. This makes the book indispensable for amateur and professional historians.

The book is designed by Marnie Soom and typeset in New Caledonia and MetroNova. The fonts are very legible, and the illustrations are excellent.

Despite all the above, I confess I have several gripes with the book design and typesetting.

First, the paper size is 10.5″ by 8.5″. This wide book is difficult to read except when sitting at a desk, and even more difficult to leaf through. The book is typeset in three columns with rather narrow margins. More generous margins could have been used for the notes, which are now put at the end of each chapter.

I also do not understand why the book is typeset ragged right. Narrow unjustified columns produce a strange look since the sizes of the gaps are visually comparable to the column width. With the ragged right margin, the indented first line of a paragraph sometimes looks centered (an example of such a paragraph can be found in an illustration above). Interestingly enough, justification was one of the crucial problems for the early Linotype machines, solved only when the company bought John Rogers's firm with its patent for the double-wedge justifier. As discussed in the book, before this purchase Linotype Co. even tried to refuse to pay Mergenthaler his royalties unless he invented a way to circumvent this patent, arguing that his machine was useless otherwise.

Last, but not least, it is not a good idea to break the line between "Mr." and the person's last name, as it is sometimes done in this book.

Despite these minor gripes it is a very good book. I think it might be especially interesting for a TEX audience. Some of the metaphors used in the description of TEX's algorithms come from the world of hot-metal typesetting. It gives a different perspective to read about the mechanical justifiers, or the literal setting of the slug. Many digital fonts we use now originated at Linotype, and it is fascinating to see how they were created.

I also think this book is essential reading for anybody interested in the history of typesetting and fonts.

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

BachoTEX 2014 proceedings

The BachoTEX 2014 proceedings was published by GUST, the Polish language TEX user group (gust.org.pl). The web page for the conference program is gust.org.pl/bachotex/2014/program.

Jean-Michel Hufflen, What can typography gain from ePub?; pp. 5–12

We show that ePub — a well-known format for electronic books — has integrated many features related to nice typography in comparison with other formats such as HTML5. However, due to their respective designs, ePub does not reach the same quality as (LA)TEX, even if some effects are easier to implement. We explain why.

Jean-Michel Hufflen, Managing name conflicts and aliasing with MlBibTEX; pp. 13–16

When several bibliography database (.bib) files are used to build a LATEX document's references, BibTEX signals an error if a bibliographical key is used more than once. A possible solution consists of renaming bibliographical entries, but MlBibTEX now provides a cleaner way, by means of namespaces associated with .bib files. Symmetrically, we can now express that a unique bibliographical item is referred by several keys. These new features are put into action by means of both MlBibTEX and an additional LATEX 2_ε package.

Jean-Michel Hufflen, Musical symbols in the digital age; pp. 17–24

First, we briefly review the musical symbols available in Unicode and MusiXTEX. Then we show why the definition of such symbols is difficult if we aim to express the whole information included in a musical score.

Taco Hoekwater, MetaPost development update; pp. 25–26

MetaPost 1.9 enabled the decimal arbitrary precision system. Using that as an example, adding the binary arbitrary precision system was a simple operation. What this means is that MetaPost 2.0 is finally finished. A momentous occasion for me, but it is also the proper moment to call it quits. I have no plans for further developments to MetaPost, nor does it seem even remotely likely that I would be able to find the time to implement any such even if I did come up with something. This will be my final MetaPost presentation as the active maintainer. The development of MetaPost will continue with Luigi Scarso as the primary maintainer.

Taco Hoekwater, Lua & TEX tokens; p. 27

LuaTEX has had a token Lua library since the early beginnings, but it was more a proof of concept, and has never worked really well at that. This talk presents a new, better interface between Lua code and the TEX language parsing.

Przemysław Scherwentke, Trup w każej szafie (o książce: LATEX dla matematyków) [A skeleton in every closet (a book review: LATEX for Mathematicians)]; pp. 28–30

The book presents the basics of typesetting with LATEX for lay people proudly called mathematicians. Its positive side is that basic packages useful for the daily tasks are presented, particularly those for typesetting tables, pictures and mathematical formulas. Unfortunately over and over again this book makes the impression of being *written* by lay people. Truths are mixed with semi-truths and false statements, and a good part of the presented examples defies the rules of proper typesetting. The article (review) lists the most notable errors along with correction proposals, and compares each to a version from a different manual. There are also a few suggestions as to what should have been given but was omitted from nearly 300 pages of the book.

Piotr Bolek, Maria Bolek, and Mikołaj Topicha-Dolny, Technika i estetyka książki elektronicznej [The technology and aesthetics of electronic books]; pp. 31–38

What are e-books? Classification of e-books; Formats of e-books; Text formatting in e-books; Devices and software for e-book reading vs. aesthetic and technical aspects; Paging and hyphenation; Typography and graphics; Interaction, animation and multimedia; Practical examples.

Piotr Bolek, Używanie fontów systemowych w TEXu w różnych systemach operacyjnych [Using system fonts with TEX in various operating systems]; pp. 39–42

A presentation for beginners and medium advanced users. The aim is to give simple prescriptions on how to use the fonts given with the operating system or other OTF and TTF fonts employing the facilities available with the modern implementations of TEX (XETEX, LuaTEX). Using alternative glyphs and activating OTF features with ConTEXt and LATEX.

Kees van der Laan, PSlib.eps Catalogue, preliminary and abridged version; pp. 43–95

A selection of PostScript definitions collected in my PSlib.eps library and documented as an e-book

catalogue is presented. Now and then variant pictures have been included from `pic.dat` which comes with `Blue.tex`. Old Metafont code has been included which may be useful for MetaPost programmers. Variants of pictures enriched by postprocessing in Photoshop show other possibilities. Escher's doughnut is a teaser which has to be done in MetaPost. Along with `PSlib.eps` is the file `PDFsfromPSlib`, which contains the pictures in `.pdf` format. The complete `PSlib.eps`, `PDFsfromPSlib` as well as the catalogue as an e-book, will be released on occasion of NTG's 25th lustrum which will be celebrated in the fall of 2014, on `www.ntg.nl`. A prerelease will be offered to GUST's file server. The (static) library for TEX standalone pictures, `pic.dat`, packaged with `Blue.tex`, will be redistributed as well.

Hans Hagen, Lua in MetaPost; pp. 96–104

For some years I have been wondering how it would be to escape to Lua inside MetaPost, or in practice, in MPlib in LuaTEX. The idea is simple: embed Lua code in a MetaPost file that gets run as soon as it's seen. In case you wonder why using Lua code makes sense, imagine generating graphics using external data. The capabilities in Lua to deal with that are more flexible and advanced than in MetaPost. Of course we could generate a MetaPost definition of a graphic from data but often it makes more sense to do the reverse. I finally found time and reason to look into this and in this article I will describe how it's done.

Hans Hagen, LuaTEX 0.79; pp. 105–108

Around version 0.50 the general picture of LuaTEX became more or less clear. Between versions 0.50 and 0.75 the program reached a level that made it possible to use it for production. Currently we're moving toward version 0.80. This version has some new features and existing features have been improved. The backend code is somewhat better separated due to a partial re-implementation of expansion. We're stepwise making the code base leaner, meaner and cleaner (again as a by-product of a critical edition project). What started as a transition from WEB to readable CWEB (an effort not to be underestimated) hopefully will become a coherent set of files with proper documentation. As there is still a long list of items to do; it will take us a few years to get there, but we're optimistic about the end goals.

In this talk I will discuss the work that has been done in the last year and present some of our plans for future versions.

Paweł Łupkowski, Online LATEX editors and other resources; pp. 109–112

(Reprinted in this issue of *TUGboat*.)

Luigi Scarso, Experiments with OpenMP and LuaTEX; pp. 113–138

This paper describes some experimental parallel functions implemented using the OpenMP API. A parallel version of `sort` is shown and discussed, and also some results about performance and efficiency.

Krzysztof Pszczoła, Książka papierowa na rynku zdominowanym przez publikacje elektroniczne: mniej powinno znaczyć lepiej [Paper books on the market dominated by electronic publications: fewer should mean better]; pp. 139–142

I am proposing a different viewpoint on the supposed evolution of the trends in connection with electronic and paper publications. The point I will make is that proliferation of electronic publications will enforce a change the way paper publications will evolve: there will be fewer of them, better edited and visually refined and, perhaps, they will bear less similarity to the books as we know them now. I envisage that, contrary to the prevailing belief that proliferation of electronic publications is a threat for people preparing paper publications (as less books will be printed), this would mean an opportunity for them (because paper publications will be better prepared and so require more work).

I will present possible business models which editors could employ when delivering both paper and electronic versions of the same content. As a brief digression, I will present a short elaboration on the possible physical forms "new books" might have (e.g., folder, leporello (concertina), or poster).

−∗ − ∗ − ∗−

The following presentations do not appear as articles in the proceedings, but have slides linked from the on-line program. All the slides are located on this page: `www.gust.org.pl/bachotex/2014-pl/presentations/`; however, there are no links from this page, so the name of the PDF file as shown below must be entered as part of the URL.

Patrick Gundlach, Using LuaTEX the hard way: How to use the internal node structure of LuaTEX to create a PDF document without using \backslashes

With LuaTEX it is possible to access the internal data structures (so-called "node lists") that TEX creates after parsing the user's input. You can analyze and modify the data before it gets written to the PDF file. It is even possible to programmatically create your own node lists and render these in the output file. One can also create a node list and instruct TEX to break the list into lines, hyphenate it or insert ligatures.

This presentation gives an introduction to how node lists work. The `lua-visual-debug` package for LuaTEX serves as an example for using the necessary callbacks to analyze the node lists and to manipulate them. A simple node list creator shows how to construct an hbox and use TEX's line breaking algorithm to get nicely formatted text.

`gundlach-1-b2014.pdf`

Patrick Gundlach, speedata Publisher: Create complex documents from databases

TEX and LATEX are well suited for many different kinds of documents, not only when formulas are needed. Once you need complex tables and lists, or an index or table of contents or when you need automatically correct cross references and bibliographies, there are few programs better suited for typesetting tasks. But there are remain many cases where LATEX has its difficulties. For example: complex tables that are broken across multiple pages with changing headers and footers and running sums; good looking paragraphs with absolutely no overfull boxes; typesetting on a grid; free (exact) positioning of objects on a page or on a page grid; using arbitrary fonts; automatically adjusting paragraph shape based on image shapes (text flows around images); using text containers with overflow; safe usage of escaping characters/catcodes; automatic selection of master pages; and more.

`gundlach-2-b2014.pdf`

Hans Hagen, What makes using TEX and MetaPost interesting

While working with TEX and MetaPost I often run into interesting situations. Sometimes they result in special styles (that probably go unnoticed), they result in additional features (that probably never get used because we forget about them), and they could also result in tracing features (which probably seldom get used). I use this opportunity to discuss a few of them that came up last year: realtime metafonts (or: a way out of lack of symbols); juggling nodes (or: pseudo-extensions to Lua/TEX); surprising side effects of hashing (or: how LuaJITTEX can be slower than LuaTEX); generating graphics (or: visualizing data).

`bachotex-2014-metapost.pdf`

Bogusław Jackowski, Piotr Strzelczyk, and Piotr Pianowski, On the progress of the TEX Gyre Math project: TG Schola Math

Three fonts — TG Pagella Math, TG Termes Math, and TG Bonum Math — have been released so far within the frame of the TEX Gyre Math Fonts project. We'll present the next font, i.e., TEX Gyre Schola Math which completes the TEX Gyre Math Font project. Of course, the maintenance of the TEX Gyre collection will continue.

`tgm-final03web.pdf`

Andrzej Tomaszewski, Cuneiform script — a phenomenon of civilization

I will talk about the evolution of this form of writing from picture to alphabetic forms. About writers, written document types and writing materials used by the people of Sumer, Babylon and Assyria.

`pismo-klinowe.pdf`

Andrzej Tomaszewski, Absolutely non-computer and completely not programmable new book forms

I will talk about the quest for new and unconventional forms of books, created chiefly in designer circles connected to artistic books and the new world trend called bookart and amongst creators of the so-called liberature. It will be an apology of Krzysztof Pszczoła's prophecies preaching the development of printed book forms.

`formy-ksiazki.pdf`

Ulrik Vieth, An improvised talk about the state of OpenType math fonts

In this talk, we review the state of OpenType math fonts which have been under development in the last few years. We discuss how to evaluate or test the quality of the design and implementation of these fonts. While a lot of progress has been made providing first releases of several new fonts, we suggest areas where additional work may be needed for improving and fine-tuning these fonts to reach production quality.

`conf-talk-ot-math-state.pdf`

[Received from Jerzy Ludwichowski and Tomasz Przechlewski.]

Die TEXnische Komödie 4/2014–1/2015

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 4/2014

Harald Lichtenstein, Mit LATEX-Bordmitteln ein eigenes Verzeichnis definieren [Defining one's own lists of document pieces with LATEX's built-in means]; pp. 21–24

A few LATEX commands are sufficient to define one's own lists similar to the table of contents or the list of figures. This article shows how.

Jens Knispel, LATEX für Psychologie [LATEX for psychology]; pp. 25–29

The goal of this article is to show the advantages of LATEX concerning scientific papers in psychology. Utilizing comparisons with MS Word, we discuss the handling of layout, tables, graphics and references as well as templates for scientific papers.

Clemens Niederberger, LATEX und Chemie [LATEX and chemistry]; pp. 30–45

According to the author's subjective sense the number of LATEX users among chemists is (very) slowly increasing. The may be due to the fact that using packages such as `chemfig`, `mhchem`, `chemmacros`, `chemnum`, `modiagram`, `endiagram` and `bohr` a chemist has numerous options to typeset his papers. This article gives an overview of the most important packages and briefly describes their functionality.

Uwe Bieling, Erweiterung des Artikels «Briefumschlädge beschriften und frankieren» [Extension of the article "How to stamp and print envelopes with LATEX"]; pp. 46–54

This article shows how envelopes can be printed and stamped for serialized letters. It is an extension of the article published in DTK 3/2012, p. 50.

Herbert Voss, Symbole [Symbols]; pp. 54–63

The `fontawesome` package is an interface for the symbol font of the same name, which is only available in OpenType format and thus can only be used with XeLATEX or LuaLATEX. The glyphs of the font are easily accessible via Xavier Danaux's `fontawesome` package.

Petra Rübe-Pugliese, CTAN sucht Mitstreiter [CTAN seeks supporters]; pp. 64–67

The CTAN world and the underlying workflows are no mystery anymore. One or two additional supporters would greatly ease the current team's workload and improve the reliability of the system.

Gerd Neugebauer, CTAN spricht Deutsch: Sprachunterstützung für das Web-Portal [CTAN speaks German: Language support for the web portal]; pp. 67–72

[Also published in *TUGboat* 35:3.]

Die TEXnische Komödie 1/2015

Thomas Hilarius Meyer, TEXnik im Wolkenkuckucksheim? Webbasierte LATEX-Editoren in Überblick [TEXnic in Cloud-Cuckoo-Land? An overview of web-based LATEX editors]; pp. 10–17

The big hype about cloud computing seems to be over but today browser-based LATEX editors allow more or less convenient editing of documents independent of time and space — and a local TEX installation. Several platforms are introduced here.

Christine Römer, Von LATEX mit LATEX2RTF zu EPUB [From LATEX to EPUB with LATEX2RTF]; pp. 18–25

This article shows how one can convert LATEX to EPUB based on LATEX2RTF and a RTF2EPUB converter. In addition the original TEX can be converted to PDF with the standard TEX workflow. This workflow is now much easier than using Docbook.

Uwe Ziegenhagen, Größere Dokumente mit LATEX erstellen [Creating larger documents with LATEX]; pp. 25–29

LATEX is well-known for its capability to handle larger documents of all kinds, since the underlying platform is very stable. But how does one organize a project with hundreds of pages? From the experience of a PhD-thesis, I want to explain how the work can be simplified.

Uwe Ziegenhagen, Die neue `scrletter` Umgebung in KOMA-Script [The new `scrletter` environment in KOMA-Script]; pp. 29–31

Since version 3.15, KOMA-Script not only offers letter functionality in the form of its own class, but also as a package. In this article I briefly explain how this feature can be used.

Rolf Niepraschk and Herbert Voss, Ausgabe einer Liste der installierten Pakete unter TEX Live [Printing the list of installed packages under TEX Live]; pp. 32–33

Sometime it is desirable to get an overview of all installed packages of a TeX installation. For TeX Live this is possible using `tlmgr info -only-installed`. The generated list, however, is not given in a format suitable for LATEX. Using a short script one can transform the provided data and execute the corresponding LATEX run.

[Received from Herbert Voß.]

TUG Business

TUG 2015 election report

Nominations for TUG President and the Board of Directors in 2015 have been received and validated.

For President, two individuals have been nominated: Kaveh Bazargan and Jim Hefferon. Thus, an election ballot is required, in accordance with the TUG election procedures (`tug.org/elecproc.html`), and has been mailed. This year, voting online is also allowed, through the TUG members area, `https://www.tug.org/members`.

For the Board of Directors, the following individuals were nominated:

Pavneet Arora, Barbara Beeton, Karl Berry, Susan DeMeritt, Michael Doob, Cheryl Ponchin, Norbert Preining, and Boris Veytsman.

As there were not more Board nominations than open positions, all these nominees are duly elected for the usual four-year term. Thanks to all for their willingness to serve.

Terms for both President and members of the Board of Directors will begin with the Annual Meeting. Congratulations to all.

Board members Taco Hoekwater, Ross Moore, Steve Peter, and Philip Taylor have decided to step down at the end of this term. On behalf of the Board, I wish to thank them for their service, and for their continued participation until the Annual Meeting.

Statements for all the candidates are appended, both for President (order determined by lot) and for the Board (in alphabetical order). They are also available online at the url below, along with announcements and results of previous elections.

◇ Kaja Christiansen
for the Elections Committee
`http://tug.org/election`

Kaveh Bazargan

(Candidate for TUG President.)

About me

I fell in love with TEX in 1983, after discovering Don Knuth's "TEX and MetaFont". I was the first TEX user in Imperial College, London, and the first to submit a PhD written in TEX. In 1988 I set up Focal Image Ltd (now River Valley Technologies) to deliver TEX typesetting to publishers. 25 years on, we are one of few typesetters paginating exclusively with TEX. I feel I am well connected in the publishing industry and want to use that influence to promote TUG.

How I have supported TUG

- Hosted and funded two TUG meetings and donated back to TUG much of the registration fees.
- Personally recorded 7 TUG meetings (and 5 local TUG meetings) and arranged post-processing and hosting, gratis — TUG 2014 is in progress. The recordings have generated 100,000s of views on `zeeba.tv`.
- Since 2004, I have worked on a TEX GUI to attract non-TEXies, including funding a free open version.

My plans for the future of TUG

Were I to be honored with the TUG presidency, here are some suggestions for building on TUG's outstanding reputation:

Increase institutional memberships and revenue

- Attract more publishers and typesetters to be members. Currently Springer is the only major publisher who is an institutional member.
- Broader pricing options: Reduce the entry to institutional membership, but increase the top rate, possibly linked to turnover.
- Offer a prominent badge for their web sites, e.g. "Supporting TUG" for institutional members. This will help publishers win support of TEX authors.

Educate the publishing industry

- TEX is a headache for publishers and typesetters. Sadly, the "industry standard" for handling TEX submissions is to convert them to Word! TUG can play a part in organizing TEX workshops which are badly needed.

Attract younger members

- Update the TUG home page with a modern "responsive" theme, and extend the current functionality.
- Add a modern front-end to CTAN, with thumbnails for each style file, graphical browsing, etc.
- Commission page designers to create non-technical templates for brochures etc.

Increase TUG meeting attendance

- Include sessions aimed at the publishing industry.
- Set up one-day conferences for publishers, through TUG, or via local TUG groups.

- Stream conferences live. Ironically this seems to increase attendance at conferences. (River Valley can offer this through `Zeeba.tv`.)

Jim Hefferon

(Candidate for TUG President.)

Statement: TEX and friends are widely used today, part of the infrastructure of mathematics and science around the world. But we have challenges. Things change so quickly in this area and we must be sure to keep our tools sharp, so that they still do the job that users need the tools to do.

TUG must continue to try to understand the needs of our members and of the entire community, and to see how we can help with those needs. We must promote the TEX suite in wider areas and to a new generation of users. And, crucially, we must take what steps we can to address the decline in our organization's membership.

As in the past, key is coordinating with our partners in other user groups around the world, continuing to hold conferences and to publish *TUGboat*, continuing to sponsor development including that of TEX Live, and continuing to help fund new projects where feasible. But we must in addition try to find new ways to make TEX and friends more visible, and to make TUG membership as attractive as possible.
Biography: I am a mathematics professor at Saint Michael's College in the US. I first extensively used TEX in the early 90s when I wrote a freely-available textbook. Some people may know me from my time working on CTAN. I am also a long-time TUG Board member and am now Vice President.

Pavneet Arora

I design and engineer control and automation systems for residential and commercial projects such as lighting, audio/video, HVAC, security.

One great shortcoming in such projects is the lack of adequate documentation — caused in no small part because few systematic methods exist with which to capture either the design or the implementation specific information. As such, my current research interests involve the specification driven documentation of signals using the TEX family of tools. I firmly believe that TEX toolsets distinguish themselves not only for the beauty of the output they produce, but also by the ease with which they integrate into modern documentation workflows as well as their ability to handle the demands placed upon them.

I am also passionate about mathematical literacy in young children, especially in cases where these students are shunted out of the mainstay curriculum, and continue to explore the development of dynamically generated math worksheets using TEX to aid in their learning.

Barbara Beeton

Biography: For TEX and the TEX Users Group:

- charter member of the TEX Users Group; charter member of the TUG Board of Directors;
- *TUGboat* production staff since 1980, Editor since 1983;
- Don Knuth's "TEX entomologist", i.e., bug collector, through 2014;
- TUG committees: publications, bylaws, elections;
- liaison from Board to Knuth Scholarship Committee 1991–1992.

Employed by the American Mathematical Society:

- Staff Specialist for Composition Systems; involved with typesetting of mathematical texts since 1973; assisted in initial installation of TEX at AMS in 1979; implemented the first AMS document styles; created the map and ligature structure for AMS cyrillic fonts.
- Standards organizations: active 1986–1997 in: ANSI X3V1 (Text processing: Office & publishing systems), ISO/IEC JTC1/SC18/WG8 (Document description and processing languages); developing the standard ISO/IEC 9541:1991 Information technology — Font information interchange.
- AFII (Association for Font Information Interchange): Board of Directors, Secretary 1988–1996.
- STIX representative to the Unicode Technical Committee for adoption of additional math symbols, 1998–2012, with continuing informal connections.

Statement: Once again I've decided it's not quite yet time to retire. TEX continues to provide interesting problems to work on, and TUG still provides a focus for dedicated TEX users.

I believe there's still a place in the TUG ranks for one of the "old guard", to provide institutional memory when it's appropriate, and cheer on the younger folks who are trying new things.

With support from the members of this wonderful community, I'd like to continue for four more years.

Karl Berry

Biography: I served as TUG president from 2003–2011 and was a board member for two terms prior to that, and one term subsequently. I am running again for a position on the board.

I have been on the TUG technical council for many years, and co-sponsored the creation of the TEX Development Fund in 2002. I'm one of the primary system administrators and webmasters for the TUG servers, and the production manager for our journal *TUGboat*.

On the development side, I'm currently the editor of TEX Live, the largest free software TEX distribution, and thus coordinate with many other TEX projects around the world, such as CTAN, LATEX, and pdfTEX. I developed and still (co-)maintain Web2c (Unix TEX) and its basic library Kpathsea, Eplain (a macro package extending plain TEX), GNU Texinfo, and other projects. I am also a co-author of *TEX for the Impatient*, an early comprehensive book on plain TEX, now freely available. I first encountered and installed TEX in 1982, as a college undergraduate.

Statement: I believe TUG can best serve its members and the general TEX community by working in partnership with the other TEX user groups worldwide, and sponsoring projects and conferences that will increase interest in and use of TEX. I've been fortunate to be able to work on TUG and TEX activities the past several years, and plan to continue doing so if re-elected.

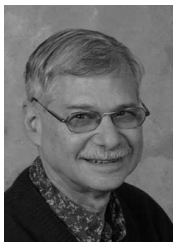
Susan DeMeritt

My name is Susan DeMeritt, I live in Lakeside, California, a suburb of San Diego.

I have been employed by the Center for Communications Research, La Jolla, in San Diego, California for almost 22 years now as the only employee in the Publications Department; I perform the technical typing duties required as well as serving as a resource for other employees with questions regarding the usage of LATEX. I started the position learning TEX and am now working with LATEX 2$_\varepsilon$. I continue to enjoy using LATEX 2$_\varepsilon$ to typeset mathematical and scientific papers; there is always something new to learn and always another challenge to figure out.

I have been a member of the TEX Users Group since 1989. I have been a member of the Board of Directors since March of 1998, and Secretary since 2001. I really enjoy being part of the Board of Directors of the TEX Users Group.

Michael Doob



I have been using TEX for more than a quarter-century. In 1984 I wrote one of the first books in pure mathematics to be printed using TEX and camera-ready copy. In those pre-laser printer days, the output used a dot-matrix printer (at a glorious 240dpi using my home-written device driver). It was entitled *Recent Results in the Theory of Graph Spectra*, and the book, the printer, and the device driver have all happily disappeared in the mists of bygone days.

TEX, on the other hand, has had an amazing evolution. It has not only developed as an elegant piece of software, but its syntax has become a *lingua franca* for many scientific fields. The basic engine has driven many applications that have revolutionized mathematical publishing among other areas. Watching these changes has been exciting and exhilarating. These applications continue to evolve and set new standards in many unexpected ways. For example, beamer has become the standard for many types of mathematical presentations.

The TEX Users Group has done a wonderful job of supporting the variations on the theme of TEX: there are useful annual meetings with interesting presentations, there are the publications *TUGboat* and PracTEX which appeal to both novice and expert, and there is support on the web using CTAN in general and TEX Live in particular. These efforts are spearheaded by the Board of Directors. I believe I can bring to this Board a background that will

facilitate its efforts. I have experience as a mathematician, as the founder of the TEX publishing office for the Canadian Mathematical Society, and as a former Board member. I would appreciate the support of you, the members, and, if elected, will give my best efforts to encourage the wider and more varied uses of TEX.

Cheryl Ponchin

My name is Cheryl Ponchin, I am employed at the Center for Communications Research in Princeton. I have been typesetting mathematical papers using (LA)TEX since 1987.

I have been a member of the TEX Users Group since 1989 and a member of the TUG Board since March of 1998. I have done many workshops for TUG as well as at several universities. I really enjoy being part of TUG.

Norbert Preining

Biography: I am a mathematician and computer scientist living and working wherever I find a job at a university. After my studies at the Vienna University of Technology, I moved to the Tuscany area of Italy for a Marie Curie Fellowship. After another intermezzo in Vienna I have (temporarily?) settled in Japan since 2009, currently working at the Japan Advanced Institute of Science and Technology, working on intermediate logics and algebraic specification languages.

After years of being a simple user of (LA)TEX, I first started contributing to TEX Live by compiling some binaries in 2001. In 2005, I started working on packaging TEX Live for Debian, which has developed into the standard TEX package for Debian and its derivatives. During EuroBachoTEX 2007, I got (by chance) involved in the development of TEX Live itself, which is now the core of my contribution to the TEX world. Up till now I am continuing with both these efforts.

Furthermore, with my move to Japan I got interested in its typographic tradition and support in TEX. I am working with the local TEX users to improve overall Japanese support in TEX (Live). In this way we managed to bring the TUG 2013 conference for the first time to Japan.

More details concerning my involvement in TEX, and lots of anecdotes, can be found at the TUG interview corner and my web site, `preining.info`.
Statement: After many years in the active development, I want to take up more responsibility by becoming a board member of TUG. In my eyes, TUG is the most influential user group, and its involvement in the development, promotion, and support of the whole TEX micro-cosmos is of essential importance for the future survival.

The challenges I see for TUG in the next years are the increase of members and funds, and technical improvement of our software. Promoting TEX as a (self-)publishing tool also outside the usual math/CS environment will increase the acceptance of TEX, and by this will hopefully bring more members to TUG.

Boris Veytsman

Biography: I was born in 1964 in Ukraine and have a degree in Theoretical Physics. I worked for various scientific and high-tech employers in Ukraine and US: universities, research companies, government contractors, etc. I participated in many different projects: from the design of industrial vacuum cleaners to the research in the thermodynamics of complex systems to the development of Internet in space to the design and implementation of air traffic surveillance system to medico-biological research — as well as teaching and writing. My CV is available at `http://borisv.lk.net/cv/cv.html`.

I have been using TEX since 1994 and have been a TEX consultant since 2005. I published a number of packages on CTAN and papers in *TUGboat* & Prac. TEX J.
Statement: As a TUG Board member I am interested in making TUG more useful for the members and attracting new members. We need this to ensure the long term relevance of our group for the community.

I arranged some discounts for TUG members from publishers and other vendors. I convinced Google to join TUG as an institutional member. Recently I helped to organize a membership drive for TUG.

If the TUG community allows me to serve this next term, I am going to continue this activity.

TUG financial statements for 2014

Karl Berry, TUG treasurer

The financial statements for 2014 have been reviewed by the TUG board but have not been audited. As a US tax-exempt organization, TUG's annual information returns are publicly available on our web site: `http://tug.org/tax-exempt`.

Revenue (income) highlights

Membership dues revenue was down about 3% in 2014 compared to 2013. (TUG began a membership campaign to try to attract new members, `http://tug.org/membership`.) Product sales were nearly doubled, primarily due to a single large Lucida site license. Contributions were slightly down. The annual conference had a large margin, due to better-than-budgeted attendance. Interest and advertising income were slightly down. Overall, 2014 income was up 7%.

Cost of Goods Sold and Expenses highlights, and the bottom line

Payroll, *TUGboat*, DVD production, postage, and other office overhead continue to be the major expense items. Most were less than budgeted; overall, 2014 COGS was down about 10% from 2013, while general expenses were down slightly.

The "prior year adjustment" compensates for estimates made in closing the books for the prior year; in 2014 the total adjustment was positive: $423.

The bottom line for 2014 was positive: almost exactly $14,000.

Balance sheet highlights

TUG's end-of-year asset total is up around $17,000 (8%) in 2014 compared to 2013.

The Committed Funds are administered by TUG specifically for designated projects: LaTeX, CTAN, the TeX development fund, and others. Incoming donations have been allocated accordingly and are disbursed as the projects progress. TUG charges no overhead for administering these funds.

The Prepaid Member Income category is member dues that were paid in earlier years for the current year (and beyond). Most of this liability (the 2014 portion) was converted into regular Membership Dues in January of 2014.

The payroll liabilities are for 2014 state and federal taxes due January 15, 2015.

Summary

TUG remains financially solid as we enter another year.

TUG 12/31/2014 (vs. 2013) Revenue, Expense

	Jan – Dec 14	Jan – Dec 13
Ordinary Income/Expense		
Income		
Membership Dues	91,785	94,800
Product Sales	13,529	7,498
Contributions Income	8,776	9,126
Annual Conference	8,720	
Interest Income	425	625
Advertising Income	390	410
Services Income	671	3,493
Total Income	124,296	115,952
Cost of Goods Sold		
Membership Drive	256	
TUGboat Prod/Mailing	18,703	24,850
Software Production/Mailing	3,076	3,038
Postage/Delivery – Members	2,294	2,923
Lucida Sales Accrual B&H	5,993	2,875
Member Renewal	406	417
Total COGS	30,728	34,103
Gross Profit	93,568	81,849
Expense		
Contributions made by TUG	2,000	3,324
Office Overhead	13,134	12,121
Payroll Exp	64,752	64,486
Interest Expense		94
Total Expense	79,886	80,025
Net Ordinary Income	13,682	1,824
Other Income/Expense		
Prior year adjust	423	194
Other Expenses	106	
Net Other Income	317	194
Net Income	**13,999**	**2,018**

TUG 12/31/2014 (vs. 2013) Balance Sheet

	Dec 31, 14	Dec 31, 13
ASSETS		
Current Assets		
Total Checking/Savings	201,400	186,696
Accounts Receivable	2,655	180
Total Current Assets	204,055	186,876
TOTAL ASSETS	**204,055**	**186,876**
LIABILITIES & EQUITY		
Liabilities		
Committed Funds	30,838	27,711
Administrative Services	1,920	4,879
Deferred contributions	45	90
Prepaid member income	7,610	4,550
Payroll Liabilities	1,094	1,100
Total Current Liabilities	41,507	38,330
TOTAL LIABILITIES	**41,507**	**38,330**
Equity		
Unrestricted	148,546	146,529
Net Income	14,002	2,017
Total Equity	162,548	148,546
TOTAL LIABILITIES & EQUITY	**204,055**	**186,876**

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*

diacriTech, *Chennai*, *India*

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*

Marquette University, Department of Mathematics, Statistics and Computer Science, *Milwaukee*, *Wisconsin*

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*

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 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.

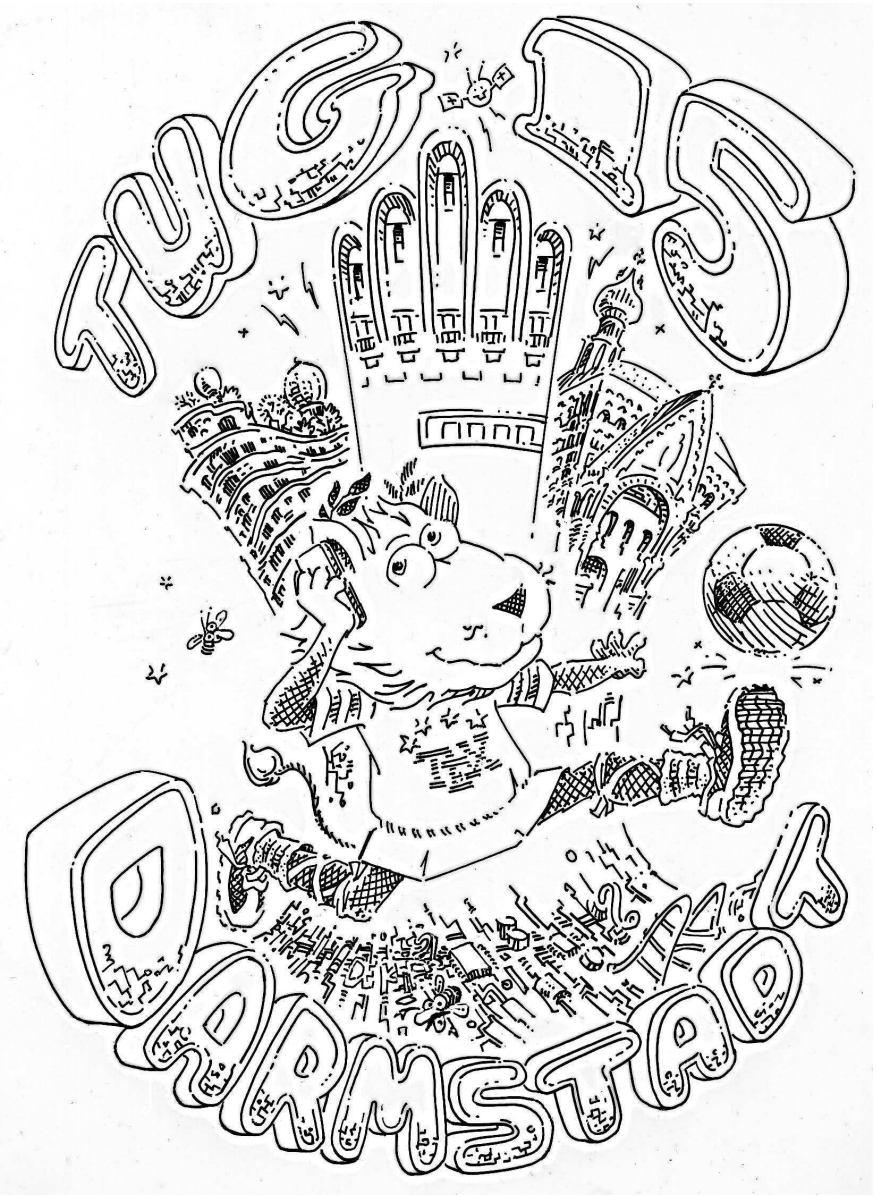
Sofka, Michael
8 Providence St.
Albany, NY 12203
+1 518 331-3457
Email: michael.sofka (at) gmail.com
Skilled, personalized TEX and LATEX consulting and programming services.

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.



The 36th Annual Meeting of the TEX Users Group

July 20–22, 2015

Welcome Hotel
Darmstadt, Germany

http://tug.org/tug2015 ■ tug2015@tug.org

April 10 — bursary application deadline
May 1 — presentation proposal deadline
May 15 — early bird registration deadline
June 1 — preprint submission deadline
July 20–22 — conference
July 31 — deadline for final papers for proceedings

Sponsored by the TEX Users Group and DANTE e.V.

Calendar

2015

Apr 10	**TUG 2015** deadline for bursary applications. tug.org/tug2015
Apr 16 – 19	DANTE Frühjahrstagung and 52nd meeting, Stralsund, Germany. www.dante.de/events.html
Apr 17 – 19	Crafting Type introductory type design workshop, Lesley University, Boston. craftingtype.com
Apr 29 – May 3	BachoTEX 2015: 23rd BachoTEX Conference, "Various faces of typography". Bachotek, Poland. www.gust.org.pl/bachotex/2015
Apr 30 – May 1	TYPO San Francisco, Yerba Buena Center for the Arts, San Francisco, California. typotalks.com/sanfrancisco
May 2 – 3	Paul Moxon — Vandercook Workshop, Museum of Printing, Andover, Mass. museumofprinting.org/txp/events2
May 11	**TUG 2015** deadline for presentation proposals. tug.org/tug2015
May 11	**TUG 2015** deadline for early bird registration. tug.org/tug2015
May 21 – 23	TYPO Berlin 2015, "Character", Berlin, Germany. typotalks.com/berlin
Jun 1	**TUG 2015** deadline for preprints for printed program. tug.org/tug2015
Jun 11 – 13	Ladies of Letterpress, "Type on the Cob", Mt. Pleasant, Iowa. www.letterpressconference.com
Jun 29 – Jul 3	Digital Humanities 2015, Alliance of Digital Humanities Organizations, "Global Digital Humanities", Sydney, Australia. dh2015.org
Jun 30	**TUG 2015** deadline for discounted hotel reservations. tug.org/tug2015
Jul 7 – 10	SHARP 2015, "The Generation and Regeneration of Books". Society for the History of Authorship, Reading & Publishing, Longueuil/Montreal, Canada, www.sharpweb.org

TUG 2015
Darmstadt, Germany.

Jul 20 – 22	The 36th annual meeting of the TEX Users Group. Presentations covering the TEX world. tug.org/tug2015

Jul 31	*TUGboat* **36**:2, submission deadline (proceedings issue).
Aug 9 – 13	SIGGRAPH 2015, "Xroads of Discovery", Los Angeles, California. s2015.siggraph.org
Aug 9 – 14	Balisage: The Markup Conference, Washington, DC. www.balisage.net
Aug 12 – 16	TypeCon 2015: "Condensed", Denver, Colorado. typecon.com
Aug 24 – 28	SHARP 2015, Society for the History of Authorship, Reading & Publishing, Jinan, Shandong Province, China, www.sharpweb.org
Sep	DANTE Herbsttagung and 53rd meeting, TU Graz, Austria www.dante.de/events.html
Sep 8 – 11	ACM Symposium on Document Engineering, Lausanne, Switzerland. www.doceng2015.org
Sep 14 – 18	9th International ConTEXt Meeting, "Taming ConTEXt", Nasbinals, France. meeting.contextgarden.net/2015
Oct 14 – 17	Association Typographique Internationale (ATypI) annual conference, Theme: "Challenges", São Paulo, Brazil. www.atypi.org
Oct 19 – 20	The Thirteenth International Conference on Books, Publishing, and Libraries, University of British Columbia, Vancouver, Canada. booksandpublishing.com/the-conference-2015

Status as of 20 March 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.

A combined calendar for all user groups is online at texcalendar.dante.de. Check the archives of this list tug.org/pipermail/tex-meetings/ for the latest postings.

Other calendars of typographic interest are linked from tug.org/calendar.html.

TUGBOAT Volume 36 (2015), No. 1

Introductory

Intermediate

Intermediate Plus

Advanced

Contents of other TEX journals

Reports and notices