TUGboat

Volume 32, Number 1 / 2011

TEX Users Group

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

Memberships and Subscriptions

2010 dues for individual members are as follows:

- Ordinary members: $95.
- Students/Seniors: $55.

The discounted rate of $55 is also available to citizens of countries with modest economies, as detailed on our web site.

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 is $100 per year, including air mail delivery.

Institutional Membership

Institutional membership is a means of showing continuing interest in and support for both TEX and the TEX Users Group, as well as providing a discounted group rate and other benefits. For further information, see `http://tug.org/instmem.html` or contact the TUG office.

TEX is a trademark of the American Mathematical Society.

Addresses

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

Telephone

+1 503 223-9994

Fax

+1 206 203-3960

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`

Have a suggestion? Problems not resolved?
The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: May 2011]

Printed in U.S.A.

The ideal proofreader should be, in my opinion, knowledgeable about every aspect of spelling, punctuation, and grammar, while being slightly dyslexic.

Isaac Asimov, *It's Been a Good Life* (2002)
edited by Janet Jeppson Asimov

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 32, NUMBER 1 • 2011
PORTLAND • OREGON • U.S.A.

TUGboat

This regular issue (Vol. 32, No. 1) is the first issue of the 2011 volume year. No. 2 will be another regular issue, with reprints from the EuroBachoTEX conference, and No. 3 will contain papers from the TUG 2011 conference in Trivandrum, India.

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 an issue's 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

The deadline for receipt of final papers for the next issue is June 30, and for the proceedings issue is October 31.

As always, suggestions and proposals for *TUGboat* articles are gratefully accepted and processed as received. Please submit contributions by electronic mail to `TUGboat@tug.org`.

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. More details and tips for authors are at `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.

Other TUG publications

TUG is interested in considering additional manuscripts for publication, such as manuals, instructional materials, documentation, or works on any other topic that might be useful to the TEX community in general.

If you have any such items or know of any that you would like considered for publication, send the information to the attention of the Publications Committee at `tug-pub@tug.org`.

TUGboat advertising

For information about advertising rates and options, including consultant listings, write or call the TUG office, or see our web pages:
`http://tug.org/TUGboat/advertising.html`
`http://tug.org/consultants.html`

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.

METAFONT is a trademark of Addison-Wesley Inc.
PostScript is a trademark of Adobe Systems, Inc.
TEX and $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX are trademarks of the American Mathematical Society.

From the President

Karl Berry

Election

After four terms as TUG president, I decided to run for director this year. Long-time board member Steve Peter expressed his desire and willingness to serve as president, and I fully support him.

As this *TUGboat* was in preparation, the election results came in, and there were no contested positions. The official board changes will not occur until October, but I take this opportunity to welcome Michael Doob and Taco Hoekwater to the board. See the election item in this issue for full details.

Conferences

The TUG 2011 conference will take place in Trivandrum, Kerala, India, hosted by River Valley Technologies: `http://tug.org/tug2011`. The deadline to receive abstracts is July 15, the early bird registration discount ends August 1, and the conference takes place October 19–21. (And thanks to Namboodiri of RVT for the conference drawings used throughout.)

Other upcoming conferences: the fifth ConTEXt user meeting in Porquerolles, France, Sept. 19–24 (`http://meeting.contextgarden.net/2011`); and TEXperience 2011 in Zelezna Ruda, Czech Republic, Sept. 28–Oct. 2 (`http://striz9.fame.utb.cz/texperience`). EuroBachoTEX 2011 will have passed by the time this issue is mailed; we expect to reprint selected articles from that conference in the next issue of *TUGboat*.

One final note on the 2010 TUG conference: a report by Dave Walden has been published as "User Group Celebrates Major Anniversary of TEX" in the *IEEE Annals of the History of Computing*, vol. 33, no. 1, January–March 2011, pp. 78–79.

Software

As I write this, work toward the TEX Live 2011 release is well underway, with most of the major program updates (LuaTEX, MetaPost, DVIPDFM*x*, ...) already committed to the source tree. We have begun trial builds and are aiming to freeze updates in June. We plan to deliver the final images to manufacturing in July. The editors of the MacTEX, proTEXt, and CTAN components are also preparing their respective releases.

Interviews

Since my last column, Dave Walden has interviewed Boris Veytsman, Herb Schulz, Malcolm Clark, and Norbert Preining for the TUG Interview Corner (`http://tug.org/interviews`).

The book of interviews we published in 2009 is available at `http://tug.org/store/texpeople`. The full PDF for the book is also available in the TUG members area.

About the cover

For the cover of this 100th issue of *TUGboat*, we made a selection of previous notable or unusual covers, from volume 1, issue 1 in 1980 (upper left), to this issue #100 (lower right). (We arbitrarily ended the recursion after one level; the PDF file size was already getting out of hand!) Most of the drawings are by Duane Bibby, but by no means all — several of the conference drawings were done by local artists. As it turned out, there were exactly 32 such covers.

At the center of it all is, of course, Barbara Beeton in full *TUGboat* work mode. She has been the editor of this journal since volume 4, issue 2, and shows no signs of slowing down. The drawing here was made by Duane and presented to Barbara at the TUG 2003 conference. (A companion drawing was made for Mimi Burbank at the same time; it is included here a couple of pages further on.)

Thanks to Robin Laakso, Stephen Moye, Mare Smith, and Dave Walden, as well as the crew at Cadmus (*TUGboat*'s printer for many years now), for their help with putting together this special cover.

In memoriam

Sadly, we've had news since my previous column of two members of our extended community passing away: Randy Kobes and Mimi Burbank.

Randy died on September 18, 2010, at his home in Manitoba, Canada. He set up the `http://mirror.ctan.org` redirector for CTAN, following (much more extensive) work he did for the Perl archive, CPAN, and for GNU (`http://ftpmirror.gnu.org`). I had the pleasure to work with him on the CTAN and GNU setup, among other small projects, and was always impressed with his attention to detail and willingness to spend time to improve the user experience. He will be missed. (Aside: the CTAN and GNU redirectors remain available, now hosted at `savannah.gnu.org`.)

Mimi died on November 28, 2010, in Uganda. She was the backbone of *TUGboat* production for many years, among many other TEX and TUG efforts. It was my pleasure to correspond with her extensively as I got up to speed with *TUGboat*; her unique personal style always made my day. I'll leave the full memorial to the other reminiscences we are printing in this issue; suffice it to say, she will be greatly missed.

◇ Karl Berry
`http://tug.org/TUGboat/Pres/`

Editorial comments

Barbara Beeton

Opus 100[1]

To my surprise and delight, *TUGboat* has reached its 100th issue, and this just after the celebration of TEX's 2^5th birthday — both causes for great celebration. In this column, I shall set down some musings about the past, and a bit of speculation on the future.

As an aside, Dick Palais, the principal instigators for the adoption of TEX at the AMS, and the first Chair (the office that preceded President) of TUG, turns 80 this year. A celebration in honor of the event, a conference on Geometry and its Applications, is announced at `www.math.uci.edu/~scgas/Palais-birthday/`. Happy birthday, Dick!

TEX has become firmly established in the math and physics community. LATEX (with many variations) has become the workhorse for production of books and journals by a number of scientific societies and some commercial publishers as well. TEX algorithms have been adopted for use by other text processing software — line-breaking in InDesign, and the math approach in the implementation of math tables for OpenType fonts. ConTEXt has made its mark on interactive on-line educational materials as well as on paper. And pdfTEX and LuaTEX now provide features not available from the original engine. This is the visible state of things.

There *are* occasional attempts to make TEX more visible. Karl's column cites the report by Dave Walden to the IEEE Computer Society, and representatives of the various European TEX groups regularly participate in gatherings of GNU/Linux and similar groups. There is an active TEX linguistics discussion forum, and at least some recognized typographers acknowledge TEX's existence when asked. But these are all specialists, and thus exceptions.

In these pages we have reported a number of "undercover" uses, where the batch capabilities of (LA)TEX have been leveraged to generate various forms from databases or interactive entry by users who are completely unaware that TEX is involved. An example of the latter, quite different from the "form" concept, was called to my attention by William Adams: `www.customstoriesinc.com/`, based on XETEX. This is the likely growth area for TEX as I see it — TEX won't disappear, it will just go further underground.

[1] Title stolen from an admired author, Isaac Asimov: `en.wikipedia.org/wiki/Opus_100`. I still remember fondly the reception I got at his book signing.

I hope someone will look back in 2042 — on TEX's 2^6th birthday — to see whether this prediction comes to pass.

BBVA award for Don Knuth

The BBVA Foundation in Madrid has announced that Don will be one of the recipients of a Frontiers of Knowledge Award, in the area of Information and Communication Technologies. The announcement on their web page reads, in part,

> U.S. scientist Donald E. Knuth takes the award for making computing into a science. His *The Art of Computer Programming* systematizes the way that human beings talk to machines and is considered the seminal work on computer science.

For complete information about the 2010 awards, see `www.fbbva.es/TLFU/tlfu/ing/microsites/premios/fronteras/index.jsp`.

The BBVA Foundation "expresses the engagement of financial group BBVA with the cause of social responsibility in the societies where it does business, in pursuit of a better quality of life for all citizens." The awards program "seek[s] to recognize and encourage world-class research and artistic creation, prizing contributions of lasting impact for their originality, theoretical significance and ability to push back the frontiers of the known world." The 2010 awards will be presented in Madrid on June 6.

Short takes

The occasion of Matthew Carter's MacArthur Fellowship has resulted in an open season on articles and interviews. In *The Economist* he is billed as "the most-read man in the world": `www.economist.com/blogs/babbage/2010/12/doyen_type_design&fsrc=nwl`. An interview with *imprint* on "How do you design a great typeface?" is shared by *Salon* at `http://shar.es/HtKeu`.

In the past, fonts tended to be taken for granted, just part of the landscape. With a computer on nearly everyone's desk or lap, they are coming out of the shadows. The Museum of Modern Art in New York has added a number of typefaces to their permanent collection and mounted an exhibition on their place in modern design, as reported here: `observatory.designobserver.com/feature/standard-deviations-types-and-families-in-contemporary-design/26428/`.

Handwriting, although it is becoming more and more a "thing of the past", is still found to be worthy of study — especially past forms: `nationalarchives.gov.uk/palaeography/`.

Most of these news bits were reported in the TYPO-L discussion list. It's a fairly low traffic list, populated by an interesting bunch of type observers and managed by Peter Flynn, who has just launched a discussion on the pros and cons of ragged right setting for dissertations. Subscription requests can be sent to TYPO-L-subscribe-request@LISTSERV.HEANET.IE.

The scientific publisher Springer has launched a new resource for mathematical typesetting: www.latexsearch.com. This site is populated with several million LaTeX code snippets containing mathematical text, drawn from their numerous publications. Much of the material seems not to be edited for quality of the LaTeX code; it is unclear whether all examples are exactly as input by authors, or might have been constructed with the "help" of various front-end software. In any event, if one wants to find uses of particular symbols or constructs, here is a place to look.

We should also mention that Don Knuth has found some errors in the book prepared for TUG's 2^5 anniversary. A list is posted on the TUG web site, at tug.org/store/tug10/errata.html; if you don't have a copy of the book in which to mark the errors, the book can be obtained as described at tug.org/store/tug10, and members can read it online from a link on that page.

Finally, it's not so often that a popular tune is animated entirely by typography. "Shop Vac", animated by Jarrett Heather, can be viewed at vimeo.com/17419652. Delightful, both the interpretation and the use of visual allusions.

Mimi

The passing of Mimi Burbank is reported by several other friends in this issue, and is the subject of several memorials on the web, among them www.portaltotheuniverse.org/blogs/posts/view/85412/ by Thilina Heenatigala, coordinator of the Global Astronomy Month and the Astro Book Drive, and kasesestreetkids.blogspot.com/2010/11/sad-news-mimi-has-died.html by some of the individuals involved with the effort to create an organization for rescuing and educating the many orphans and street kids in western Uganda.

Here I would like to reminisce a bit on what Mimi meant to me.

By 1995, the year in which the annual meeting was held in St. Petersburg, Florida (with Mimi in charge of arrangements), *TUGboat* was falling behind schedule with longer and longer delays (a consequence in part of concentrating too much responsibility in the hands of one person). Mimi stepped forward and offered her site for production and archiving. She rescued our vessel! From then until her retirement in 2005, Mimi was our production manager — always there when needed, a tactful but determined nag, more concerned, it seemed, about the responsibility she had taken on than her own immediate interests. In other words, a good and devoted friend.

For the most part, our communications were by e-mail; we had met earlier only at a few TUG meetings, and we hadn't really had a chance to just sit down and talk. In 1996, we got that opportunity, traveling together to attend the meeting in Russia — and used the time well! We were met at the Moscow airport by Irina Makhovaya, who most generously accommodated us in her apartment until it was time to proceed to Dubna, and gave us a personal tour of Moscow. It was on this trip that I learned how readily Mimi connected with people. She made a number of friends among the Russian attendees, and remained in touch long after we had returned home.

When she decided to retire, Mimi dutifully offloaded all the *TUGboat* backlog from the SCRI computer at Florida State by moving it to the TUG box in Denmark, and turned it over to Karl. But she didn't disappear. Oh, no! Her communications just changed from *TUGboat*-related topics to news about her new home and friends in Kasese, Uganda. Her stories about goats (she admitted to a weakness for them) were believable only because they came from Mimi.

We last corresponded just a few days before her death. Although she had suffered some health setbacks, and a shipment of her medications "disappeared" as it came through customs, she kept her sunny outlook and showed her Ugandan family how an American Thanksgiving was celebrated. A few days later, we received the news from her son Charles that his mother had passed away.

Mimi was a person who made a difference. I miss her greatly, but treasure the honor and privilege it was to have known her.

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

Mimi Burbank

Jackie Damrau

Mimi Burbank and I met during one of my early TEX User Group conferences in the mid-1980s. We even shared a room at Stanford during that TUG conference and served on the Board of Directors together. We became instant friends as we were both engaged in our careers working for scientific laboratories or university departments. During my ten years as a TUG member, Mimi was always enjoyable to be with, encouraged me to give my best to the Group, and to step outside of my comfort zone. After my departure from TUG as my career moved from the use of LATEX to other desktop publishing software, Mimi and I stayed in touch. As the years slipped away, though, we lost track of each other.

Yet, Mimi was doing a Google search one day and found one of my old TUG conference proceedings papers. She emailed me; we talked for a while on the phone and got caught up on our life stories. That was not too long ago when she was working on a TUG conference proceedings.

My fondest memories of Mimi were that she always had a great smile and funny laugh and her penchant for giving to and serving the TEX User Group. She advocated the use of TEX even in her semi-retirement years. Mimi was a wonderful person that will be truly missed.

◇ Jackie Damrau
STC
jackie dot damrau (at) comcast dot net

Missing Mimi

Christina Thiele

My last exchange with Mimi Burbank was Nov. 19, 2010. I'd seen a CTAN notice about a new package, excel2latex — it just sounded so neat, that I thought she'd like to see it. Turned out that she actually *needed* something just like it:

> I've got lots of files I'd love to add to my latex files - but can't because of the excel spreadsheet junk... but now it looks like I can ;-) may have to give it a try...

And the very next day:

> attached is a PDF file - not finished of course, but I'm so glad I've got this pkg now...
> need to remember how to do landscape and all sorts of things... and need to read some more directions - but this file is one I have to send to donors in the UK all the time, and doing it
> using a google email "docs" page is not my favorite way to do things....
> so glad you sent the the info on it...
> mimi

Typical!

* * *

Ten days later, on the 29th, I got a call from Charles Burbank — Mimi Burbank had died the night before, in Kasese, Uganda, at the age of 69. She was buried there three days later, in St. Paul's Cathedral cemetery. She had two sons, Charles and Bill. And a world of friends.

* * *

I've been struggling to write this piece, which Karl suggested a few months ago, and I think it's because, for me, Mimi's not really 'gone'. She lives on, in a way that could never have been imagined (much less possible) 30 years ago — she 'lives' on the 'net, in web pages and in e-mail that's spread around the entire planet.

She's a permanent presence on my computer, in yearly folders of e-mail I've kept, both personal and TUG-specific, that date back to 1991. I've been re-reading some of our exchanges and they make me smile, or laugh outright. The woman had a sense of humour that would always, at the end of the day, make any short-lived storm revert to its proper dimensions as a tempest in a teapot — or a real problem which needed to be addressed head-on. The sheer quantity of mail also reminds me that, during the '90s, we didn't seem to have anything in our lives other than our jobs and TUG :-).

And that's just between us two! My machine is only one small home where SCRIMING MIMI can be found — she's present in so many more places.

Sometimes our exchanges would get lost on her monitor, as she'd have over a dozen windows open on her computer, both work and correspondence in progress, on-site and off, working on issues ranging from system or software or administrative matters, to TEX questions and TUG work. I have to say, Mimi never met a question she didn't want to find an answer to, either from her own extensive experience or someone else's! And what she didn't know, she'd find someone to solve it. Or she'd figure it out herself. She was tireless when it came to getting a job done ... getting many jobs done!

Indeed, one of Mimi's greatest qualities was her lack of fear — learning new programs, installing new software, working with new hardware ... whereas I have always met 'the new' with varying degrees of hesitation and reluctance. I've always admired that quality in her, and continue to remind myself of it.

* * *

Mimi and I both started attending TUG meetings in 1987, but I didn't actually meet her until the 1991 Dedham meeting in Massachusetts.[1] I was on the Program Committee, and had been asked to join Hope Hamilton's editorial team as well, for the proceedings.[2] Along with Dian De Sha from CalTech, Mimi and I were her editorial assistants, each of us given articles to work on. Working through our assigned articles, we all got to know one another via e-mail, and Mimi and I took off like a house on fire, as they say. It was that collaborative work on the proceedings that cemented our friendship, and she subsequently joined TUG's board in 1993 (I'd been a member since 1988). And it was Mimi who initiated an as-yet unpublished article on the whole experience, entitled "A Tale of 4 Witties", which I'd in fact just re-sent her in mid-October of last year, to her great amusement.

We had both become intensely interested in TUG's conference proceedings and program committees, and eventually were part of the move to expand the production team approach to regular *TUGboat* issues. A great stimulus for this was Mimi's having found a way to arrange that SCRI, her work site (the Supercomputer Computations Research Institute in

[1] We continued attending together until the 1995 meeting in Florida, which we jointly oversaw; she went to one more, in Russia in 1996; and my last one was the 1999 Vancouver meeting.

[2] I'd been proceedings editor for both the 1988 and 1989 meetings; Lincoln Durst was editor for the one in 1990, and Hope and I had met when lending him a hand with it.

Tallahassee, Florida), provide computer space and access for team members to work remotely.[3] We had Robin Fairbairns, Sebastian Rahtz, and Michel Goossens over in Europe, and then Barbara, Mimi, and me in North America — we were like a round-the-clock service bureau :-).

Our deep involvement with TUG activities carried on throughout the '90s, gradually returning to just the *TUGboat* production team by the start of the new millennium. During my term as TUG president, I leaned heavily upon her support and advice, both of which always evinced common sense laced with humour.

Gradually, though, we each eased out of our TUG roles, me by 2000 and Mimi by 2005. Mimi had become increasingly active in her church, St. Peter's Anglican Church, in Tallahassee, which culminated in her decision to leave the States permanently, for St. Peter's mission in Uganda. Now *that* was a big surprise — Mimi, whose first trips outside the country had been the Aston meeting in 1993 (we drove around Wales for a week after that!), the 1996 Dubna meeting in Russia, and a week's vacation here in Ottawa in August of 1998, deciding to *move* permanently to Uganda. Wow!

Before she left, Mimi wrote up her TUG experiences in an interview with Dave Walden for TUG's Interview Corner. One of her non-TUG activities that should be mentioned is that of providing on-line support for Y&YTEX, along with Robin and myself, in the early 2000s. The list we started, yandytex, is now on the TUG server, having moved there after Y&Y finally ceased operations.

* * *

Within weeks of having left for Uganda, Mimi was again on-line with correspondence to everyone 'back home' — she'd just changed jobs, that's all! The stories had changed, from systems and software and administrative matters, to the travails of her car, the beauty of the countryside, and the generosity of its people, who fast became her new family. And Mimi herself was being transformed, into Mama Mimi, who applied all her skills not just from SCRI and TUG, but from even before then, when she'd gone to nursing school. And what she didn't know, she'd search for, via the web, to hammer away at a life that was so different from the States, and so much more in need.

Many of us would send her packages with items she said were desperately needed, books and school supplies above all. I'd fill in the spaces with a few things for her, too — seeds, spices she couldn't find locally, toiletries, even some candies from time to time :-). In return, Mimi would send us photos and stories about so many daily events. She even joined in the fun of a school project my daughter had, of sending a 'Flat Stanley' cut-out to Kasese, and snapping photos of the 'cardboard boy' in various locations around town.

She would write these great, long messages to us (sometimes there were over 60 cc's on such mail!), and then finally decided to pull them all into a journal structure — using pdfTEX, in fact. She also set up a web page at the St. Peter's site, remotely, from Uganda (!), as well as working on the local site for the South Rwenzori Diocese. Eventually she became a Skype user, much to her son Charles' delight.

Mimi went back to the States a couple of times, for health reasons, and finally these became too much for her. She died in the night of Nov. 28, having turned 69 on the 24th and celebrated Thanksgiving on the 25th.

Of the many causes, be they formal or simply benevolent, Mimi felt passionate about, I know of two: BUFO (Base Camp United Christian Foundation), and the Astro Book Drive. A donation to either would be a fitting memorial to this best of friends, most constant of colleagues — a single person whose life has not ceased to make a difference.

◇ Christina Thiele
15 Wiltshire Circle
Nepean, ON K2J 4K9, Canada
cthiele (at) ncf dot ca

References and resources

1. Interview Corner, TUG web site: www.tug.org/interviews/burbank.html
2. Mimi's website and journals: www.saint-peters-archives.org/files/mamamimi/
3. Mimi's Facebook page: www.facebook.com/people/Mimi-Burbank/1063046798
4. Articles about Mimi's work in Kasese, Uganda: www.liquida.com/kasese kasesestreetkids.blogspot.com/2010/12/burial-of-dear-mimi.html
5. St. Peter's Anglican Church, Tallahassee: www.saint-peters.net
6. South Rwenzori Diocese: www.southrd.org
7. Base Camp United Christian Foundation and contact person: www.basecampngo.org
8. Astro Book Drive and contact person: astrodrive.lakdiva.net and bit.ly/f8M3Gt
9. Mailing list for the Y&Y TEX system: lists.tug.org/yandytex

[3] Reading the Production Notes for issues 15:2 through 16:1 will provide a quick snapshot of the fundamental changes that were wrought in the latter part of 1994 and into 1995.

16 years of ConTEXt

Hans Hagen

1 Introduction

When Karl Berry asked me to wrap up something for the 100th issue of *TUGboat* I didn't hesitate too long to agree. Of course you then end up with the dilemma of what to write down and what to omit, but it's worth a try.

When you're asked to look back it is sort of unavoidable to also look forward. In this article I will reflect on some of ConTEXt's history and spend a few words on its future. First I will try to describe the landscape in which we have ended up.

2 Perceptions

After being present for some 16 years, about half the lifespan of the TEX community and its *TUGboat*, there has been enough published about ConTEXt to give the reader at least an impression of what it is about. However, to some it might (still) be confusing, especially because for a long time in TEX publicity, 'TEX' was viewed as nearly equivalent to LATEX. In itself this is somewhat troublesome, because TEX is a generic system but such is the situation. On the other hand, nowadays banners for conferences, cartoons and other promotional material mention multiple engines, MetaPost, LATEX and ConTEXt, fonts and more, so the landscape is definitely more varied.

Over the past decades I have run into descriptions of ConTEXt that are somewhat curious and they give a good impression of what ConTEXt users run into when they have to defend their choice.

- "It is a package that can be loaded in LATEX." This perception is natural for LATEX users as packages are part of their concept. On the other hand, for ConTEXt users, the package concept is alien as we have an integrated system. A quick look at the way ConTEXt is embedded in the TEX directory structure will learn that it does not relate to LATEX and I'm sure that a simple test will show that loading it as a style will fail.
- "It is a variant of plain TEX and has similar concepts, capabilities and limitations." For sure there are a couple of commands that have the same name and similar functionality but there it stops. We used to load plain TEX as a base because it provides some infrastructure, but even there most was overloaded. On the other hand, we feel that when a user reads *The TEXbook*, he or she should not get an error on each command that is tried, especially not math.
- "It is meant for and depends on pdfTEX." Actually, we started out using DVI and when we switched to using outline fonts we used DVIPSONE. Other DVI backends were also supported, but when pdfTEX came around it was pretty convenient to have all functionality in one program. Maybe because we were involved in pdfTEX development, or maybe because ConTEXt always supported the latest PDF tricks, this myth arose, but all functionality was always available for all backends.
- "It is best used for presentations." It is a fact that at user group meetings, the presentation aspect was quite present, if only because I like making new styles as part of preparing a talk. However, it is just a matter of styling. On the other hand, it has drawn some users to ConTEXt.
- "I don't see you using math and you're discussing features that I never needed, so why use TEX at all?" This comment was given after I gave a presentation about ConTEXt doing MathML where I explained that I liked content markup more than presentational markup.
- "I've been present at talks but only recently realized that you were talking about an integrated macro package that is independent of other packages." This kind of remark is of course an indication that I'd forgotten to explain something. It also says something about TEX marketing in general.
- Some comments are baffling, like "I saw you talking to Frank. Isn't that competition?" As far as I know there is no competition, maybe different audiences at most. The community is large enough for multiple solutions. And most of all, we don't sell anything and I always try to keep my own (commercial) work separated from presenting ConTEXt.
- "We don't need this, we're satisfied with the way we do it now." Remarks like that occasionally come up when someone presents something new. I don't care too much about it myself because in most cases I know that I can achieve more with TEX than the person making such a remark, but it can be very demotivating for those doing their first presentation.

I'm sure that ConTEXt is not alone in getting such comments. I remember that there have been discussions about embedding the PDF backend into the TEX engine and strong argument for keeping it separated. Especially when arguments come up like "We should keep TEX as it is", which boils down to "We should not change Don's work", it gets nasty. It

is a fact that the DVI backend is just an example of a backend and there can be many others. The same is true for the `\pdfsomecommand` extensions: deep down they use whatsits and these are definitely meant for extensions. Any argument for using specials exclusively is wrong as specials themselves are an extension.

Right from the start Don Knuth made clear that extending TEX and friends was part of creating solutions. When Volker Schaa and I presented the Latin Modern fonts to Don Knuth, one of his first comments was "Why didn't you fix the mu?". He could not do it himself because Computer Modern is as frozen as pure TEX, but that does not mean that it cannot be done elsewhere! It's interesting to hear users defend a status quo while those they think they are defending definitely keep moving on. I'm sure that I don't speak for myself alone when I say that Don Knuth is one of the main reasons why I've chosen the TEX route and keep following it. It makes TEX and its community special in many ways.

The reason for mentioning this is that when you are part of the flow of developments around TEX, you also have to deal with conservatism. In itself this is understandable as TEX is a tool that you will use forever once you've typeset more than a few documents with it. And there is a nice aspect worth mentioning here: as pure TEX will always be there, and as derived engines are as closely as possible downward compatible, you can stick to 25-year-old methods and customs as long as you keep your macros and fonts around! No one is forcing you to update or switch to another engine or use another macro package. And you can still produce the same perfect output as years ago. The best proof of that is the author of TEX himself, and you can bet that he knows pretty well what developments happen around TEX and friends.

In the following story I will mention some of the design decisions and reasons for going the ConTEXt route. I will use two qualifications there: ConTEXt MkII, the version meant for pdfTEX and XETEX, and ConTEXt MkVI, the new version for LuaTEX. (More about these in the last section.) The MkVI variant is a complete rewrite and as part of the process I threw away lots of code that I had always before considered to be final. Code that I spent weeks or more perfecting, and that evolved along with getting more experienced in the macro language, code that has been optimized to the max, code that I got emotionally attached to because I know for what occasion I wrote it. It gets frozen into MkII and is never used again by myself, but it can be run forever anyway. That's what TEX is about: accumulating experiences. In a few weeks I will travel to BachoTEX again. There, among TEX friends, it will be clear once more that we're still moving forward, that a 30-year-old TEX is not yet ready for retirement, even if some of its first time users are getting close to that.

3 Running into TEX

I always take 1996 as the year that ConTEXt showed up in public. That year sits between the two first international TEX conferences that I attended: EuroTEX 1995 in Arnhem and TUG 1997 in San Francisco. That means that this year ConTEXt is about 16 years old and as a consequence only half of all *TUGboat* publications can contain articles that refer to it.

We started using TEX a bit earlier. I still remember the bookshelves in the store where I first saw volumes A to E and because at that time I was programming in Pascal and Modula the content looked quite familiar. However, as I was neither involved in typesetting nor a mathematician it did look intriguing. Nevertheless I bought *The TEXbook*, and reading something without the ability to run the related program is somewhat special. In successive years, whenever I picked up the *The TEXbook* I was able to understand more of the neat tricks described in there.

4 The first experiments

The real reason for using TEX came when I was involved in a project where we had to get some advanced math on paper. The customer used a special typewriter for this but I remembered TEX and considered it a better tool for the job. We bought a copy of the program from Addison Wesley and got ourselves a decent printer only to find out that our customer preferred the typewriter method over typesetting.

This didn't stop us, and we decided to use TEX for typesetting our own reports and the courses that we developed along with experts in the field. I did an inventory of alternatives but they were either too expensive or closed (and obsolete within years after that moment) so in retrospect the choice for TEX was not that bad. Using TEX at that time definitely made our budgets for hardware rise: faster computers, larger disks, better and faster printers, suitable connections between them, etc.[1]

There is one thing that keeps coming back when I think about those times: we were acting in complete

[1] Before free versions of TEX came to desktops we had to actually buy TEX and friends. Interestingly I'm quite sure that it still accumulates to the largest amount of money we ever spent on software, but competing systems ran into five digit numbers so it was no big deal at that time.

isolation. There was no Internet the way there is now, and when it came our way, using it as a resource was no option with slow modems. We had no email and were unaware of user groups. The university that I had attended, and especially our department had top of the line equipment but TEX was simply unknown. We used ASCII terminals and I had written a formatter for the mainframe that helped making reports and could paginate documents: poor man's styling, pseudo-floats, tables of contents. I think that I still have the source somewhere. However, no one ever bothered to tell the students that there were formatters already. And so, when we moved on with our business we were quite unaware of the fact that something like TEX was part of a bigger whole: the TEX community.

5 Personal usage

In fact this is also the reason why the first steps towards a macro package was made. The floppies that we bought carried LATEX but the rendered output was so totally un-Dutch that I had to change files that I didn't understand at all. Of course some localization had to happen as well and when we bought an update I had to do all that again. After a while I figured out how to wrap code and overload macros in a more permanent way. Itemizations were the first to be wrapped as we used lots of them and the fact that they got numbered automatically saved us a lot of time.

Because we were involved in writing course material, we had workflows that boiled down to investigating learning objectives, writing proposals, collecting and editing content, and eventually delivering a set of related materials. It is therefore no surprise that after a while we had a bunch of tools that helped us to do that efficiently. It was only around that time that we ourselves actually profited from a TEX-based workflow. We had our own editor[2] that provided project support based on parsing structure, syntax highlighting, as well as a decent edit–view cycle.

In my job I could chair a session, drive home, sit down and wrap up the progress in a document highlighting the most recent changes and the participants would have a print on their desk next morning. The time spent on writing new macros was nicely compensated by efficiency.

We're speaking of the beginning of the nineties now. We already had dropped LATEX after a few documents and via the more easily configurable LAMS-TEX moved on to INRSTEX which was even more configurable. The fact that these variants never caught on is somewhat sad, as it indicates that in spite of TEX being so flexible only a few macro packages are available.[3] Around 1995 we had a decent shell around INRSTEX and much code was our own. I didn't understand at all what alignments were all about, so for tables we used Wichura's TABLE package and as output routines were also beyond me, we stuck to the INRSTEX page builder for quite a while. We called the beast pragmaTEX simply because we needed a name and it didn't occur to us that anybody else could be interested.

6 A larger audience

It was around that time that I became aware of user groups and we also joined the Internet. Because we had to do a lot of chemical typesetting, in particular courses for molding latex and plastic, I had written a macro set for typesetting chemical structure formulas for my colleague (who coincidentally had a background in chemistry). As there was interest for this from Germany, represented by Tobias Burnus, it was the first piece of code that went public and because we used macros with Dutch names, I had to come up with a multilingual interface. Tobias was the first international user of ConTEXt.

In the meantime I had become a member of the NTG as well as of TUG. Around that time the 4TEX project was active and it carried the first version of the renamed macro set: ConTEXt. It is at that time that Taco Hoekwater and I started teaming up our TEX efforts.

We started publishing in MAPS and *TUGboat* and after being introduced to the Polish and German user groups also in their journals. So, around 2000 we were better aware of what was happening in the larger community.

At some point I had ordered copies of *TUGboat*s but I have to admit that at that time most of it simply made no sense to me so I never really read that backlog, although at some moment I did read all Don's articles. It might be fun actually going back in time once I retire from writing macros. But the fact that there were journals at least gave me a sound feeling that there was an active community. I do realize that much of what I write down myself will not make sense either to readers who are not at that moment dealing with such issues. But at least I hope that by skimming them a user will get the impression that there is an active crowd out there and that TEX keeps going.

[2] The editor was called texedit and was written in Modula, while its follow-up, called texwork, was written in Perl/TK.

[3] Of course TEX is not unique in this: why should billions of users use only a few operating systems, editors, drawing programs or whatever?

7 How ConTEXt evolved

For this reflective article, I spent some time hunting up the past before actually sitting down to write ... here we go. The first version of ConTEXt was just a few files. There was some distinction between support macros and those providing a more abstract interface to typesetting. Right from the start consistency was part of the game:

- there were define, setup, and start-stop mechanisms
- keywords and uniform values were used for consistent control
- layout definitions were separated from content
- there were projects, products, components and environments
- the syntax was such that highlighting in an editor could be done consistently
- we had support for section numbering, descriptions and of course items
- content could be reused (selectively) and no data or definition was keyed in more than once (buffers, blocks, etc.)

As a consequence of the structure, it was relatively easy to provide multiple user interfaces. We started out with Dutch, English (the first translation was by Sebastian Rahtz), and German (by Tobias Burnus). Because Petr Sojka gave students the opportunity to do TEX-related projects, Czech followed soon (David Antos). Currently we have a few more user interfaces with Persian being the latest.

There is an interesting technical note to make here. Because ConTEXt is keyword-driven and uses inheritance all over the place it put some burden on memory. Just in time we got huge emTEX and I think in general it cannot be underestimated what impact its availability had: it permitted running a decent set of macros on relatively powerless personal computers. Nevertheless, we ran out of string space especially but since the hash was large, we could store keys and values in macros. This was not only space-efficient but also faster than having them as strings in the source. It is because of this property that we could relatively easily provide multiple interfaces. Already in an early stage a more abstract description in XML format of the interface was added to the distribution, which means that one can easily create syntax highlighting files for editors, create helpers and include descriptions in manuals.

Right from the start I didn't want users to even think about the fact that a TEX job is in most cases a multipass activity: tables of contents, references, indexes and multipass optimization means that unless the situation didn't change, an extra run is needed. It is for this reason that a ConTEXt run always is managed by a wrapper. When I realized that ConTEXt was used on multiple platforms I converted the Modula version of texexec into Perl and later into Ruby. The latest version of ConTEXt uses a wrapper written in Lua.[4] I think that the fact that ConTEXt came with a command line utility to drive it for quite a while set it apart. It also created some myths, such as ConTEXt being dependent on this or that language. Another myth was that ConTEXt is just a little more than plain TEX, which probably was a side effect of the fact that we kept most of the plain commands around as a bonus.

It was long after using LATEX that I understood that one of its design decisions was that one should write styles by patching and overloading existing code. In ConTEXt it has always been a starting point that control over layout is driven by configuration and not by programming. If something special is needed, there are hooks. For instance, for a very special section title users can hook in macros. Ideally a user will not need to use the macro language, unless for instance he or she wants some special section header or title page, but even then, using for instance layers can hide a lot of gory details.

I think that switching from one to the other macro package is complicated by the fact that there are such fundamental differences, even if they provide similar functionality (if only because publications have so much appearance in common). My impression is that where LATEX draws users because they want (for instance) to submit a paper in a standard format, the ConTEXt users come to TEX because they want to control their document layout. The popularity of for instance MetaPost among ConTEXt users is an indication that they like to add some personal touch and want to go beyond pure text.

As a consequence of consistency ConTEXt is a monolithic system. However, special areas are dealt with in modules, and they themselves are also monolithic: chemistry, MathML, presentations, etc. For a long time being such a big system had some consequence for runtime or at least loading time. Nowadays this is less an issue and with the latest and greatest MkIV we even seem to get the job done faster, in spite of MkIV supporting Unicode and OpenType. Of course it helps that after all these years I know how to avoid bottlenecks and optimize TEX code.

As I'm somewhat handicapped by the fact that in order to understand something very well I need to

[4] One can argue that this is a drawback but the fact that we use TEX as a Lua interpreter means that there are no dependencies.

write it myself, I have probably wasted much time by (re)inventing wheels. On the other hand, finding your own solution for problems that one deals with can be very rewarding. A nice side effect is that after a while you can 'think' in the language at hand and know intuitively if and how something can be solved. I must say that TEX never let me down but with LuaTEX I can sometimes reimplement solutions in a fraction of the time I would have needed with the pure TEX way.

8 Fonts and encodings

When we started with TEX a matrix printer was used but soon we decided to buy a decent laser printer (duplex). It was a real surprise to see that the Computer Modern Fonts were not that bold. Our first really large printer was an OCE office printer that was normally sold to universities: it was marketed as a native DVI printer. However, when we tried to get it running, we quickly ran into problems. By the time the machine was delivered it had become a PostScript printer for which we had to use some special driver software. Its successor has already been serving us for over a decade and is still hard to beat. I think that the ability to print the typeset result properly was definitely a reason to stick to TEX. The same is true for displays: using TEX with its font related capabilities is much more fun with more pixels.

At the time of the switch to OCE printers we still used bitmaps (and were not even aware of PostScript). First of all, we needed to tweak some parameters in the generation of bitmap fonts. Then we ran into caching problems due to the fact that each DVI file relates id's differently to fonts. It took the support people some time to figure that out and it tricked me into writing a DVI parser in Lisp (after all, I wanted to try that language at least once in my lifetime). We decided to switch to the Y&Y previewer and PostScript backend combined with outline fonts, a decision that we never regretted. It was also the first time that I really had to get into fonts, especially because they used the texnansi encoding and not the usual 7-bit TEX encoding. It must be said: that encoding never let us down. Of course when more language support was added, also more encodings had to be supported. Support for languages is part of the core so users don't have to load specific code and font loading had to fit into that approach.

In traditional ConTEXt the user can mix all kind of fonts and input encodings in one document. The user can also mix collections of fonts and have several math font setups in parallel and can have different math encodings active at the same time. For instance the Lucida fonts had a different setup than Computer Modern. The pattern files that TEX uses for hyphenation are closely related to font encodings. In ConTEXt for languages that demand different font encodings in the same document we therefore load patterns in several encodings as well.[5] Because we mostly used commercial fonts as part of MkII we provide some tools to generate the right font metrics and manipulate patterns.

The reason for mentioning all this is that a font subsystem in a TEX macro package always looks quite complex: it has to deal with math and due to the 8-bit limitations of traditional TEX this automatically leads to code that is not always easy to understand, especially because it has to suit limitations in memory, be efficient in usage and behave flexibly with respect to weird fonts. In MkIV we're using Unicode, OpenType, and wide Type 1 fonts so much of the complexity is gone. However, font features introduce new complexities if only because they can be buggy or because users want to go beyond what fonts provide.

As a side note here, I want to mention the font projects. The real reason why Volker Schaa and I took the first initiative for such a project (currently Jerzy Ludwichowski is leading the project team) is that we came to the conclusion that it made no sense at all that macro packages were complicated by the fact that for instance in order to get guillemets in French, one has to load fonts in an encoding most suitable for Polish just for these glyphs. Because in ConTEXt by default all languages are loaded and no additional special language packages are needed, this was quite noticeable in the code. What started out as a normalization of Computer Modern into Latin Modern Type 1 fonts and later OpenType variants, moved on to the Gyre collection and currently is focusing on OpenType math fonts (all substantially funded by TEX user groups). The ConTEXt users were the first to adopt these fonts, not only because they were more or less enforced upon them, but also because the beta releases of those fonts are part of the so called ConTEXt-minimals distribution, a subset of TEX Live.

9 Interactivity

The abovementioned Y&Y previewer supported hyperlinks and we used that in our workflow. We even used it in projects where large and complex documents had to be related, like the quality assurance

[5] It was one of the reasons why we moved on to patterns in UTF encoding so that users were free to choose whatever encoding they liked most. Nowadays UTF encoded patterns are standard in TEX distributions.

manuals fashionable at that time. As a result, by the time that PDF showed up, we already had the whole machinery in place to support Adobe Acrobat's interactive features. At that time PDF had two different audiences: prepress (printing) and online viewing and we were able to provide our contact at Adobe with advanced examples in the second category.

Unfortunately, with a few exceptions, none of our customers were much interested in that kind of documents. I even remember a case where the IT department of a very large organization refused to install Acrobat Reader on their network so we ended up with products being distributed on floppies using a dedicated (ASCII) hypertext viewer that we built ourselves.[6] The few projects that we used it for were also extremes: hundreds of interlinked highly interactive documents with hundreds of thousands of links. Those were the times that one would leave the machine running all night so that in the morning there was some result to look at. At that time we set up the first publishing-on-demand workflows, using either TEX input or XML.

One of the more interesting projects where interactivity came in handy was a project where we had to identify lots of learning objectives (for some 3000 courses) that needed to be categorized in a special way so that it became possible to determine overlap. With TEX we generated cross-linked dictionaries with normalized descriptors as well as documents describing the courses. It was a typical example of TEX doing a lot of work behind the screens.

Personally I use the interactive features mostly in presentations and writing a (new) style for an upcoming presentation is often the first step in the preparation. In my opinion the content and the form somehow have to match and of course one has to avoid coming up with the same style every time.[7] Maybe ebooks will provide a new opportunity, given that they get better quality screens. After all, it's a pretty trivial and brain-dead activity to produce ebooks with TEX.

10 XML

There is some magic that surrounds XML, and it is often part of a hype. I can waste pages on stories about structure and non-structure and abuse of XML and friends, but I guess it's not worth spending too much energy on it. After all, the challenges can be interesting and often the solutions come right on time, although I admit that there is some bias to using tricks you've just implemented.

Currently most of what we do involves XML one way or the other which is a consequence of the fact that ConTEXt can process it directly. As TEX is rather related to math typesetting we supported MathML as soon as it came around, and although we use it on projects, I must say that most publishers don't really care about it.

Apart from the fact that angle brackets look cool, advanced reuse of content seldom happens. This is no real surprise in a time where the content changes so fast or even becomes obsolete so that reuse is no option anyway. On the other hand, we manage some workflows for publishers that need to keep the same (school) method around for more than a decade, if only because once a school starts using it, you have to support it for some five years after the first year. In that respect it's hard to find a system that, after some initial investments, can stay around for so long and still provide occasional updates as dirt cheap as a TEX can. Unfortunately this way of thinking is often not present at publishers and the support industry happily sells them pages (each time) instead of workflows (once set up the price per page is close to zero). It does not help that (driven by investors) publishers often look at short term profits and accept paying a similar amount each year instead of paying a bit more upfront to save money later.

Maybe I'm exaggerating a bit but most projects that we run involve someone with vision on the other end of the table. Some of our customers take real risks by introducing solutions that go against the flow of time. The simple fact that TEX-based systems somehow guarantee a constant result (and at least some result) makes that succeed. Already several times we were surprised by the fact that by using TEX a solution could be provided where all previous attempts so far had failed: "This is the first automated publishing project that actually works." This might come as a surprise for TEXies who see such automation daily.

We also support companies that use ConTEXt as part of a workflow and the nice thing about that is that you then deal with experts who know how to run, update and integrate TEX. Of course specific code written for such customers finally ends up somewhere in ConTEXt so that maintenance is guaranteed.

11 Design

In the beginning we used TEX mostly in products where we were responsible for the result: no one really cared how it looked like in the end (read: no money could be spent on typesetting) so it was just

[6] In the beginning even the reader cost money so it is no surprise that it took a while before PDF took off.

[7] This is especially true when I know that Volker Schaa, one of my benchmarks in the TEX community, will be present.

an added value and we had complete freedom in design. For the last decennium we have dealt only with the rendering of whatever input we get (often XML) that no one else can render conforming to the specs of the customer. We implement the styles we need and set up workflows that can run for ages unattended. Of course we do use TEX for everything we need to get on paper, so there's also the personal fun aspect.

In that respect there is an interesting shift in usage of ConTEXt: for a long time we ourselves were the ones that drove new functionality, but nowadays it's regular TEX users that request specific extensions. So, where for us an efficient XML machinery is relevant, for users high-end typesetting in their specific field can be the focus. Of course we can do what is asked because most functionality has already been there for a while and often extending boils down to adding a key and a few lines of code. It is my impression that ConTEXt users really like to come up with a personal touch to their documents' look and feel, so fun is definitely part of the game.

Currently there are interesting developments related to the Oriental TEX project which in turn trigger critical edition support and more modern follow-ups on that kind of typesetting. Currently Thomas Schmitz is taking the lead in this area and I expect interesting challenges in the next few years.

12 The upgrade

A couple of years into this millennium I ran into a rather neat scripting language called Lua. This language is used as extension language in the SciTE editor that I use most of the time and after a while I wondered how it would be to have something like that available in TEX. As I don't touch the engine myself I asked Hartmut Henkel to patch pdfTEX into LuaTEX and after some experiments it didn't take much to convince Taco to join in: the LuaTEX project was born.

While initially just a little bit of access to some registers as well as a way to print back data to the TEX input was provided, we quickly started opening up the whole machinery. Once the potential became clear it didn't take much before the decision was made to make a special version of ConTEXt for LuaTEX. It was at the second ConTEXt user meeting that those present already agreed that it made much sense to freeze the ConTEXt for pdfTEX and XETEX and focus development on the next version for LuaTEX.

Although I have used pdfTEX for a long time (and was also actively involved in the development) I must admit that already for some years I only run it when a user reports a problem. In that respect we have already crossed the point of no return with ConTEXt. Since I never used XETEX myself, support for that engine is limited to additional font support in the MkII code and I know of some users using it, if only because they needed Unicode and OpenType while waiting for MkIV to reach a more stable state. Of course we will forever support the older engines with MkII.

Let me stress that the LuaTEX project is not about extending TEX but about opening up. Of course there are some extensions, for instance in the math engine as we need to support OpenType math, but the fundamentals are unchanged. Hard coding more solutions into the core engine makes no sense to me. First of all it's quite convenient to use Lua for that, but most of all it saves endless discussions and makes maintenance easier.

I would like to stress that the fact that most users having already switched to this version helped a lot. I'm pretty sure that the beta version is more popular than the regular (current) version. This is not only a side effect of active development, but also of the fact that the so-called minimals are quite popular. The original minimals were our self-contained, ConTEXt-only subset of TEX Live that we also put on the website, but at some point Mojca Miklavec and friends adopted it and already for some years it is the de facto reference implementation that can easily be synchronized to your personal workstation. Another important factor in the support chain is the Wiki, also known as the ConTEXt garden, an initiative by Patrick Gundlach. One of its spin-offs is Taco's additional TEX Live package repository. The minimals and garden also play an important role in providing up-to-date binaries of LuaTEX and MetaPost.

A for me quite interesting experience was that a few years ago on the ConTEXt list some users showed up who know the ConTEXt source pretty well. For a long time only Taco and I touched the code, apart from language related issues, where users sent us corrections of label translations. Most noticeable is Wolfgang Schuster. Not only is he posting many solutions on the list and writing nice modules in a style that perfectly matches the code base, but he's also amazingly able to nail down problems and I can integrate his patches without checking. Another developer worth mentioning is Aditya Mahajan. It's great to have someone in the team who knows math so well and his website `http://randomdeterminism.wordpress.com` is worth visiting. I could and should mention more, like Luigi Scarso, who is always exploring the frontiers of what is possible, or Thomas Schmitz, who not only makes beautiful presentations but also is a great tester. And

of course Willi Egger, the master of layout, composition and binding. And we are lucky to be surrounded by specialists on fonts and PDF standardization.

13 The future

So, what is the current state of ConTEXt? As we now have a complete split of the code base between traditional ConTEXt (MkII) and the new version (MkIV) we can go further in upgrading. Although one of the objectives is to be as compatible as possible, we can try to get rid of some inconsistencies and remove mechanisms that make no sense in a Unicode age. Some parts are rewritten in a more modern and flexible way and there are cases that more Lua code is used than TEX code (although of course at the Lua end we also use TEX core functionality). Also, all the tools that come with ConTEXt have been migrated to Lua. Eventually the code base will be completely redone.

In addition to coding in TEX a user can code in Lua using a user interface similar to the one in TEX, so if you know the ConTEXt commands, you can also use Lua and create so-called ConTEXt Lua Documents. At the TEX end we go a step further. Apart from some upgraded interface-related macros, for instance we have a better although somewhat less efficient inheritance model, we also support some extensions to the macro coding, like more extensive namespace support and named parameters. Files using these features are classified as MkVI. This numbering scheme is not a ratio scale — although one can argue that MkIV is twice as good as MkII, the difference between MkIV and MkVI is mostly cosmetic. It is an interval scale, so MkVI is definitely a bit better than MkIV. So for the moment let's qualify it as a nominal interval scale of numbering, one that also works out quite well in file names.

Some of the adventurous module writers (like Aditya and Wolfgang) have adopted this strategy and provide useful input to the directions to choose. It must be noted that at the time of this writing it is because of the active participation of Aditya, Luigi, Mojca, Peter, Taco, Thomas, Willi, Wolfgang and whomever I forget to mention that we can undertake such a major rewrite. On the agenda is a rewrite of code not yet scrutinized, of output routines (including various multi-column support) additional (rewritten or extended) support for tables, better access to the internal document model, an extension of the multiple stream model, maybe some CSS and DOM support, and whatever else comes up. Eventually most code will be in MkVI format. As we proceed, for sure there will be articles about it in this journal.

Of course I should mention my colleague Ton Otten, who has always been very supportive and patient with whatever I came up with. He is responsible for convincing potential customers to follow our TEX route to solutions and often he is the first to suffer from updates that for sure come with bugs. Without him we would not be where we are now.

That leaves me mentioning one person who has always been extremely supportive and open to new developments: Karl Berry. Without him you would not be reading this and I would not even have considered wrapping this up.

◇ Hans Hagen
http://pragma-ade.com

TUGboat's 100 issues — Basic statistics and random gleanings

David Walden and Karl Berry

Abstract

TUG was founded to provide an organization for people who are interested in typography and font design, particularly those interested in Don Knuth's TEX typesetting system. TUG's print journal *TUGboat*, now at its 100th issue over 32 years (certainly a noteworthy run), has been and remains an important component in carrying out TUG's mission.

1 Remembering Knuth's *3:16* book

Casting about for an appropriate article for the 100th issue of *TUGboat*, we remembered Donald Knuth's book entitled *3:16 Bible Texts Illuminated*.[1] In that book Knuth describes studying the Bible by looking at chapter 3, verse 16 of each of 59 books of the Old and New Testaments of the Bible (he left out books that were so short they don't have a verse 3:16 — if the book was long enough but stopped short of verse 16 in chapter 3, he kept counting from the last verse of chapter three into chapter four until he got to the sixteenth verse). For each such arbitrarily (randomly) selected verse, Knuth's book has four pages: (1) a sketch of the book as a whole; (2) a calligraphic transcription of the verse (each from a different renowned calligrapher), (3–4) Knuth's restatement of the verse in contemporary English and a description of his research and analysis of the verse and placing it in context.

In the book's foreword and afterword, Knuth discusses how his random sampling approach (a mathematician and computer scientist's approach) might, and in fact did, produce a revealing picture of the Bible more generally. This suggested to us that a random sampling approach might also be an interesting way to get an overall picture of the first 100 issues of *TUGboat*. Also, there would be a certain symbolism in using a method promulgated by Knuth.

2 Our random sampling approach

We first considered choosing the 100th page (symbolic of this 100th issue) of each yearly volume. However, that had the problem that a couple of volumes didn't have as many as 100 pages. Furthermore, the order of papers in issues is not random, being organized in categories, such as "Macros" and "Fonts", in an order more or less consistent from issue to issue. Always using page 100 reduces the chances of selecting a page in a category that typically appears before page 100 of the first issue of a volume.

Thus we decided to select a random page from each volume, summing the page totals on the rare occasions when an issue 2–4 started over at page 1, and for the remaining volumes using the largest page number in the table of contents (TOC) of the last issue of the year, as found in the online *TUGboat* TOCs (`http://tug.org/TUGboat/Contents`). These preparations for finding a random page in each issue immediately gave us some statistics about *TUGboat*. This is all summarized in Figure 1.

We also patched our program that generates the

vol	year	total pages	physical issues	random page	notes
1	1980	23	1	21	
2	1981	267	3	219	
3	1982	88	2	31	
4	1983	132	2	118	
5	1984	168	2	79	
6	1985	174	2	140	
7	1986	198	3	125	
8	1987	352	3	93	
9	1988	342	3	256	
10	1989	765	4	102	
11	1990	693	4	494	
12	1991	588	3	167	four logical issues
13	1992	544	4	396	
14	1993	445	4	213	
15	1994	508	4	359	
16	1995	443	4	110	
17	1996	411	4	263	
18	1997	321	4	245	
19	1998	440	4	427	
20	1999	404	4	286	
21	2000	440	4	427	
22	2001	376	3	60	
23	2002	359	3	41	four logical issues
24	2003	624	3	285	
25	2004	232	2	106	excludes TUG'04 conference preprints
26	2005	302	3	26	
27	2006	268	3	268	
28	2007	384	3	208	
29	2008	488	3	38	
30	2009	183	3	40	
31	2010	340	3	249	
32	2011	128	1	37	one 2011 issue
Total pages:		11430			
Total issues:		100			
Average pages/issue:		117			
Average pages/year:		365			

Figure 1: *TUGboat* statistics and information (average page figures are based on extrapolation of first 2011 issue size to the full year).

[1] A-R Editions, Inc., Middleton, WI, 1991.

all-*TUGboat* tables of contents and lists (see our companion article in this issue) to count the number of titles and authors across all 100 issues. The result was 3,127 titles and 2,446 authors which equals, for what it's worth, an average of 31 titles and 24 authors per issue. These numbers are generous, since they include each independent item on the TOC pages, such as the front cover, complete issue PDF, etc.

Volumes of *TUGboat* include the TUG annual conference proceedings for 1989–2010, excluding 2004 which was published as a Springer-Verlag book, three PracTEX conference proceedings, four EuroTEX conference proceedings, one combined EuroBachoTEX conference proceedings, and one NorthEast U.S. conference proceedings.

The described random selection method has the bias of leaving out unnumbered pages at the ends of some issues which at various times included TUG job opening notices, the TUG mailing list, TUG membership and ordering information, *TUGboat* submission information, TEX and METAFONT errata, the membership list, an order form for the book *Joy of TEX*, other advertisements, an $\mathcal{AMS}$-TEX panel discussion, profiles of TEX installations, *Computers and Typesetting* errata, changes and supplements, and other miscellany. However, sticking to numbered pages was easier than having to logically renumber all pages in a volume to include unnumbered pages.

Also, the largest page number listed in the online TOCs might leave out a follow-on page to the last TOC entry since article start pages and not page intervals are given in the online TOCs. The cover pages (c1–c4) were also ignored. Finally, as of issue 100 we only have the first issue of volume 32, and we did the random page selection from the 124 pages of that single issue (once we were pretty sure of the number of pages in the issue including this article).

For the computations, we used an online random number generator (`http://quantitativeskills.com/sisa/calculations/random.htm`), rather than programming, for instance, the linear congruence pseudo-random number generator described by Knuth in *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, Chapter 3, "Random Numbers" — just because it was the easier path.

The reader might well ask, "Would it not have been more symbolic to choose a random page from each of the 100 issues rather than from each of the 32 volumes, and this could also have provided a usefully larger sample?" We answer, "Perhaps, but we were writing a journal article, not a book; it would also have taken considerably more effort — though we acknowledge that deliberately choosing the less-effort approach departs from the Knuthian way."

3 The selections and notes

In this article for a single journal issue we cannot follow Knuth's model of four pages for each sample. Instead we made a copy of each article containing a randomly selected page, and then we thoughtfully (no longer randomly) made appropriate comments on the randomly selected page, the paper more generally, the author, and/or what it suggested about TUG and *TUGboat*.

Even someone who has never seen another issue of *TUGboat* may get a reasonable idea of its coverage from the following set of samples. On the other hand, long-term readers of *TUGboat* may be reminded of favorite articles, or other *TUGboat* articles not included in the random samples. We have also included notes on and references to current developments relating to the selected papers.

The randomly selected page numbers shown in column 5 of Figure 1 result in the papers below being chosen for each volume. Links to this set of selected articles may be found on the TOC page for this issue, `http://tug.org/TUGboat/tb32-1`.

1. Questions & Answers; Letters; Miscellaneous, **1**:1, p. 21.

On pages 2–3 of the first issue of *TUGboat* (the only issue in the first volume), editor Robert Welland explains, "The TEX Users Group (TUG) met at Stanford University in February of 1980 ... and among other things decided that the group would publish a newsletter to assist the distribution of TEXpertise." These pages are worth reading (`http://tug.org/TUGboat/tb01-1/tb01edit.pdf`) and include some justification for the name *TUGboat*. The first issue appeared in October 1980, and covered a variety of early TEX and *TUGboat* topics, including reports from the February users group meeting.

Our randomly selected page from that issue contains requests that committees organized at the users group meeting identify themselves and that people send reports on TEX sites, introduces two letters from satisfied users, and mentions a TEX Errata list and the TUG mailing list. The "newsletter" was up and was finding its way.

2. "TEX for the HP 3000", by Lance Carnes, **2**:3, pp. 25–26.
3. "TEX-news from Pisa", by L. Aiello and S. Pavan, **3**:1, pp. 31–32.

The randomly selected pages for both volumes 2 and 3 were in the Site Reports area (`http://www.tug.org/TUGboat/Contents/listkeyword.html#CatTAGSiteReports`), an important *TUGboat* feature for a decade or more as the TEX community spread The selected article in volume 2 covered 13

TEX sites in 15 pages, and the volume 3 report covered 5 sites in 5 pages.

Lance Carnes has remained a notable figure in the TEX community ever since, first as the "small TEX" department editor for *TUGboat* and later (and still) as the vendor of PCTEX, a well-known commercial distribution of TEX.

4. "Summary of AMS-TEX", by Michael Spivak, **4**:3, pp. 103–126.

The first issue of *TUGboat* included an article on AMS-TEX, one of the first large macro formats created for TEX. by its creator, Michael Spivak. Here, he gives an update consistent with TEX82.

5. "First principles of typographic design for document production", by Richard Southall, **5**:2, pp. 79–90.

Richard Southall was a type designer and typographer, invited to be part of the digital typography group at Stanford in the early 1980s. Of this paper he says, "Leslie Lamport and I taught a two-day course on 'First principles of typographic design for document production' as a preliminary to the TUG meeting at Stanford University, August 13–14, 1984. What follows is an expansion, somewhat revised and restructured, of my lecture notes." We can speculate that this content influenced the way that Lamport thought about the design of LATEX.

6. "Assembling a moderately-priced, high-performance clone of the IBM PC for running TEX", by M. Pfeffer and A. Hoenig, **6**:3, pp. 14–145.

Running TEX on smaller and smaller computers was an ongoing topic in *TUGboat* until personal computers became logically big. This article is the first instance of a column entitled "Typesetting on Personal Computers", and it starts at the beginning by providing an instruction manual for buying and assembling the parts of an appropriate personal computer.

7. Title page, **7**:3, p. 125.

TUGboat title pages are generally given regular page numbers in sequence, and this one was randomly selected for our sample (Figure 2). We can see that Barbara Beeton was already three years into her almost thirty year (and still counting) tenure as *TUGboat* editor. From TUG's address, we also have a hint of the major role AMS played in the early years of TEX (and continues to play, in that Barbara and other TEX developers are AMS employees).

8. Advertisements, **8**:1, pp. 87–96.

This issue of *TUGboat* included 10 pages of TEX-related advertisements. For many years TEX was a leading-edge development, and many people hoped to make money from TEX.


...books look like books because, though it sounds a bit simple to say so, that's what they are. And that's what they're supposed to look like.

Richard Hendel
"A book designer's odyssey"
Scholarly Publishing,
July 1986, p. 350

TUGBOAT

THE TEX USERS GROUP NEWSLETTER
EDITOR BARBARA BEETON

VOLUME 7, NUMBER 3 • OCTOBER, 1986
PROVIDENCE • RHODE ISLAND • U.S.A.

Figure 2: Title page of volume 7, issue 3.

9. "TEX output devices", by Don Hosek, **9**:3, pp. 251–260.

In the days before PDF, a big issue was making TEX (and DVI) work with a wide variety of output devices; thus, for a while *TUGboat* had regular reports on output devices for TEX (`http://tug.org/TUGboat/Contents/listkeyword.html#CatTAGOutputDevices`). This particular report consisted of 10 pages of charts and tables about various output devices.

10. "Contents of archive server as of 16 January 1989", by Michael DeCorte, **10**:1, pp. 97–102.

With the advent of the Internet and FTP, naturally it made sense to have online archives of content, in this case relating to LATEX. DeCorte wrote five articles from 1988–1990 on the content and organization of this archive, hosted at Clarkson University, along with providing contact information to acquire TEX material on diskettes, etc. The TEX archives spread, through collaboration and independent development, becoming CTAN today (`http://www.ctan.org`).

11. "Comments on the future of TEX and METAFONT", by Nelson Beebe, **11**:4, pp. 490–494.
11. "Editorial Comment", by Barbara Beeton, **11**:4, pp. 494–496.

There are two articles on the randomly selected page (page 494) of volume 11 — one article ending and one starting. In the context of Knuth's then-recent announcement that he had finished his work with TEX and METAFONT, Nelson Beebe discussed TUG's ongoing role, TEX's place in the world, the need for continuing TEX and TEX-related developments, and so on.

As if to emphasize the continuing viability of TEX and related activities, Barbara Beeton touched on a significant number of TUG and TEX meetings, *TUGboat* activities, etc. TEX, TUG, and *TUGboat* were continuing with or without Knuth.

12. "Some TEX manuals", by Angela Barden, **12**:1, pp. 166–170.

Author Barden critiques five early well-known books on TEX and LATEX (she likes Leslie Lamport's LATEX manual) and mentioned a couple of others. She discusses her struggle to learn TEX and summarizes her philosophy of what would make a good tutorial book on TEX. In other words, her article is indicative of a long-standing problem with TEX, that has never fully been solved, particularly given the open-endedness of the TEX world (so different from a highly specified commercial product).

13. The Donald E. Knuth Scholarship: 1992 Scholar and 1993 announcement, **13**:3, pp. 395–396.

Named for Don Knuth, the scholarship was aimed at recognizing and promoting the use of TEX by "support" personnel (as opposed to professors and programmers). Up to $2000 was provided toward attending the TUG conference. In keeping with other changes in the TEX world, the scholarship has not been awarded since 1997 (`http://tug.org/committees.html`).

14. "A format compilation framework for European languages", by Laurent Siebenmann, **14**:3, pp. 212–221.

The author developed a package to enable TEX formats (Knuth's word for a set of macros such as those that define plain TEX) to include hyphenation patterns for many languages, along with other multilingual support. Over the years *TUGboat* has published papers on many interesting ideas or experiments that never saw wide-spread use and where the perceived problem was later more or less solved by a more general capability. (In the TEX distributions of today, all suitable hyphenation patterns are included by default.)

15. "TEX innovations at the Louis-Jean printing house", by Maurice Laugier and Yannis Haralambous, **15**:4, pp. 438–443.

Over the years since the creation of TEX, various publishers, typesetters, and printers have made use of TEX, particularly as a component in automating their processes. Some of these entities have created their own tools to move between TEX and other document processing systems, and this article describes one such case.

16. "A practical introduction to SGML", by Michel Goossens and Janne Saarela, **16**:2, pp. 103–145.

The authors of this 43-page article explained, "This article discusses the basic ideas of SGML and looks at a few interesting tools. It should provide the reader with a better understanding of the latest developments in the field of electronic documents in general, and of SGML/HTML in particular." The article was one of seven articles that made up *TUGboat* **16**:2 (June 1995), grappling with how TEX would fit into the rather new world of SGML, HTML, hyperlinks, and so on.

17. "TEX in Russia: ab ovo, or About the TEXnical evolution in Russia", by Irina A. Makhovaya, **17**:3, pp. 259–264.

As TEX spread and became multi-lingual, the TEX world became regionalized to a considerable extent. This paper sums up the situation in Russia ca. 1995.

18. "Typographers' Inn", by Peter Flynn, **18**:4, pp. 242–245.

This was the first of now a dozen or so columns (still continuing) with this title, about typography as much TEX, that Peter Flynn has written for *TUGboat*. In this first installment, Flynn addressed some concerns that *TUGboat* should be exclusively focused on TEX.

19. "Hey — It Works!", by Jeremy Gibbons, **19**:4, pp. 426–427.

This column contained hints and tricks for doing things with TEX with items from a variety of people. This column had been in the separate publication *TEX and TUG News* publication (`http://tug.org/pubs.html`), and continued for four more years in *TUGboat*. The complete collection is available online at `http://tug.org/TUGboat/hiw`.

Our random selections for volumes 18 and 19 are to columns that appeared over a period of time. A few other columns that have appeared more of less regularly for a period of time are: the already mentioned "Site Reports" and "Output Devices"; Victor Eijkhout's "Bag of [macro] Tricks"; Peter Wilson's "Glisterings"; "The Treasure Chest" with

several editors over the years; and Aditya Mahajan's "ConTEXt Basics for Users".

20. "*MathKit*: Alternatives to Computer Modern mathematics", by Alan Hoenig, **20**:3, pp. 282–289.

Much activity in the TEX world has been about using other fonts than Knuth's Computer Modern set. Math fonts have been of particular interest to the community. Author Hoenig developed a tool to allow people to create their own math fonts to match existing fonts, e.g., Baskerville. Like a number of other ideas suggested in *TUGboat*, this interesting approach didn't survive the years, as more fonts included math from the outset. (At `http://mirror.ctan.org/info/Free_Math_Font_Survey` is a nearly-comprehensive survey of free math fonts currently available for TEX.)

21. "Micro-typographic extensions to the TEX typesetting system", by Hàn Thế Thành, **21**:4, pp. 317–434.

Hàn Thế Thành's success in making TEX use PDF as a native output format, in parallel with DVI, was perhaps the most important step in sustaining TEX in the world since the creation of LATEX. It was fitting that *TUGboat* publish his full Ph.D. thesis on this work as an issue of *TUGboat*. Now, a decade later, his micro-typographic extensions are finally available and commonly used in all the major TEX distributions.

22. "The status quo of the $\mathcal{NTS}$ project", by Hans Hagen, **22**:1–2, pp. 58–66.

TUGboat has often included items on various distributions of TEX, TEX engines, and so forth. $\mathcal{NTS}$ was a reimplementation of TEX in Java using an object-oriented approach that had significant support from the TEX user groups. This thoughtful article by Hans Hagen reviews the project and draws a number of interesting conclusions. We speculate that this analysis by Hans helped him sort out his own thinking about opening up and extending TEX, with resulting major developments in LuaTEX, MetaPost, and ConTEXt.

23. "FarsiTEX and the Iranian TEX community", by Behdad Esfahbod and Roozbeh Pournader, **23**:1, pp. 41–45.

Among the more difficult directions TEX has been pushed has been the desire for typesetting Persian and Arabic text.

24. Abstracts from *MAPS* 28, Fall 2002, **24**:2, pp. 283–285.

As the TEX world expanded, many other TEX user groups came into being, especially in Europe, some with their own journals, typically with all or most articles in the language of the particular country. Thus *TUGboat* was (and is) no longer the only publication with a focus on TEX and related things, and it was appropriate for *TUGboat* to include abstracts of articles from these other journals, such as these from *MAPS*, the journal of the Dutch group NTG (Nederlandstalige TEX Gebruikersgroep). A list of all such journals, past and extant, is at `http://tug.org/pubs.html`.

25. "The ℵ (Aleph) project", by Giuseppe Bilotta, **25**:1, pp. 105–107.

ε-TEX was an extension of the TEX engine, primarily implemented by Peter Breitenlohner, that added right-to-left typesetting among a variety of other new features. Omega was an effort by John Plaice and Yannis Haralambous to move TEX more into a multi-cultural, multi-lingual world, including native support for Unicode. Aleph was an effort by Giuseppe Bilotta to produce a more stable version of Omega. The lasting impact of Omega and Aleph now resides in LuaTEX, while the ε-TEX extensions were incorporated into pdfTEX and are widely used and available.

26. "Using the RPM package manager for (LA)TEX packages", by Tristan Miller, **26**:1, pp. 17–28.

Managing the plethora of (LA)TEX packages and organizing distributions has long been an issue. Author Miller proposed an approach using the RPM tools standard in the GNU/Linux world.

27. TEX consulting and production services, **27**:2, pp. 285.

A regular feature in *TUGboat* is advertisements for companies and people wishing to provide TEX-related consulting and production services for a fee. Some ads have run for many years. One can find the original ads from each issue in the whole-issue PDFs on the *TUGboat* web site (`http://tug.org/TUGboat`). (The links from each issue's table-of-contents page go to the current list of advertisers, `http://tug.org/consultants.html`.)

28. "Installing ConTEXt expert fonts: Minion Pro", by Idris Samawi Hamid, **28**:2, pp. 200–209.

Installing fonts and accessing them within TEX is another common theme through *TUGboat*'s run. These days, there are at least four domains of such font installation articles: for plain TEX, standard LATEX, XETEX and LuaTEX with their OpenType support, and the ConTEXt system. Author Hamid is also deeply involved with the current Oriental TEX project for Arabic typesetting on which he works closely with Hans Hagen, creator of ConTEXt.

29. "Do we need a font system in TEX?", by Hans Hagen, **29**:1, pp. 28–33.

Hans Hagen and Taco Hoekwater, along with a cadre of other workers, are probably pushing the opening up of TEX (and in effect contemporary rewriting) harder than anyone else. Hans is fond of intriguing and provocative titles for his papers, which are typically thoughtful and full of new ideas and wise ways of thinking about things. As Hans says in the introductory paragraph of the present article, "... working on LuaTEX and ConTEXt MkIV... gives us much time and opportunity to explore new frontiers and reconsider existing ConTEXt features."

30. "Managing bibliographies with LATEX", by Lapo Mori, **30**:1, pp. 36–48.

Managing bibliographies well is one of TEX's strong suits, thanks to the BIBTEX program developed by Oren Patashnik as part of the original Stanford TEX project. Several definitive (for the time) articles have appeared in *TUGboat* about managing bibliographies. Author Mori provided a reasonably comprehensive overview of the available methods.

31. TUG 2010 abstracts, **31**:2, pp. 248–249.

Not infrequently, presenters at TUG annual conferences decline to provide a written paper to be included in the *TUGboat* proceedings for the conference. In most of these instances, an abstract is included in its place. Fortunately, in recent years some TUG conferences have been videotaped (thanks to the efforts of Kaveh Bazargan at River Valley Technologies), and thus videos of some unpublished presentations are available on the Internet (`http://river-valley.tv`).

32. "LATEX at Distributed Proofreaders and the electronic preservation of mathematical literature at Project Gutenberg", by Andrew Hwang, **32**:1, pp. 32–38.

Our thirty-second and final randomly selected paper is a particularly appropriate "choice". It brings *TUGboat*'s reporting about TEX and the TEX world back around to Knuth's original purpose for creating TEX: the creation of well-typeset books, especially math books (a capability that was being lost as early computer typesetting systems replaced pre-computer typesetting processes). In this article, Andrew Hwang describes a project to create, using LATEX, electronic versions of classic mathematics books published in the 19th and early 20th centuries. Knuth used TEX to preserve the quality of pre-computer typesetting in newly-written math books. This part of the Distributed Proofreaders project is using TEX's high quality of typesetting in the electronic preservation of pre-computer mathematics books themselves.

4 Reflections

These random samples from 32 volumes of *TUGboat* suggest the breadth of the journal's coverage of the TEX world, while still leaving many specific categories unmentioned. `http://tug.org/TUGboat/Contents` points to a comprehensive list of topic areas used over the years by *TUGboat* to categorize its articles, and full author and title lists, as well as the issue-by-issue tables of contents. Scanning these gives an even greater sense of *TUGboat*'s diversity.

Of course *TUGboat* has served as a newsletter for the TEX Users Group (and the broader TEX world) about projects, events, and people. It has simultaneously provided tutorial material for various levels of TEX practitioners, a forum for new ideas to be suggested and experiments to be described, and a place where major new developments in the TEX world can be permanently documented. *TUGboat* articles have been peer-reviewed, but it has never been a journal of pure academic scholarship; it has served the typical role of a scientific or engineering journal in allowing participants in the field to learn about and build on (or create alternatives to) the work of others, thus extending to practice and education as well as the occasional theoretical article. Furthermore, it has played a role beyond TEX, regularly dealing with non-TEX issues of typography, design, document preparation and display.

Don Knuth has stated that computer science departments had to exist because there was a group of people who thought in a certain way which didn't fit within the confines of either traditional math or engineering departments. Perhaps because it evolved out of a creation by Knuth, *TUGboat* serves a similarly unique role for a collection of people who have interests in or needs for typography, font design, and a powerful typesetting system, and who in many cases want to go beyond the capabilities allowed to users of commercial typesetting systems. Users of TEX and related technologies are a somewhat self-selecting community, and *TUGboat* exists and is edited to serve broadly the interests of that community.

Acknowledgments

Thanks to Barbara Beeton, who edited and proofread this article, as always. Also, over its 32 years literally thousands of people have contributed to *TUGboat* as authors, columnists, editors, members of the production team, creators of the web site, and so on. The *TUGboat* vessel is a significant community of its own within the worldwide TEX community.

◇ David Walden and Karl Berry
`http://tug.org/TUGboat/Contents`

TUGboat online

Karl Berry and David Walden

1 Introduction

TUGboat has traditionally been, and remains, a print journal. Its history was described and profusely illustrated in Barbara Beeton's 2006 paper ("How to Create a TEX Journal: A Personal Journey", *TUGboat* 28:1, 2007, pp. 29–49, `http://tug.org/TUGboat/tb28-1/tb88beet-keynote.pdf`).

Since the web came into being, *TUGboat* has also existed online. This article sketches the evolution of the online version of *TUGboat*.

2 First steps

TUGboat has been part of the TUG web site since the web site's beginning. To a large extent, the original pages remain, with the basic description of the journal, editor's wish list, information for authors, and so on.

It was apparent from the outset that it would be useful to publish back issues on the web. Mimi Burbank began the process of scanning paper copies. In a few cases articles were reprocessed, or DVI converted to PDF, but in the main, sources, figures, and/or fonts were lacking, so no systematic reprocessing could be attempted. Scanning to PDF was the only viable approach.

As the work proceeded, Mimi created table of contents files in HTML by hand. The scanning took so much more time than writing the HTML, and was proceeding so slowly, there was no need to think about any tools.

As the years passed, additional volunteers came forward. In particular, Brooks Moses (with the support of his department at Stanford) ended up doing the bulk of the scanning. Robin Laakso in the TUG office also made significant contributions. Most of the issues were scanned by 2006, though the process was not completely finished until 2009, so the scanning overlapped with the other work we describe below. (Even now, additional work remains — some of the early issues were scanned at 150 dpi, some pages are crooked, a few pages are missing, etc.)

Once enough back issues had become available, the question arose of how best to make the material accessible on the web. Our attempts to answer that question are described in the rest of this article.

3 Two steps toward automating generation of *TUGboat* tables of contents

3.1 *The PracTEX Journal*

In 2004, TUG, at the suggestion of Lance Carnes, sponsored a TEX conference explicitly focused on practical use of TEX. Near the end of the conference, Tim Null and Robin Laakso discussed developing a section of the TUG web site that would contain introductory material on TEX organized by category. In an exchange of emails between Tim Null and Karl Berry after the conference, Tim came up with the alternative idea of an online journal. Lance Carnes became the editor of that journal, and other people volunteered or were recruited to join the new journal's editorial board.

The Editorial Board's planning discussions naturally were centered on issues such as the target audience, look and feel, and frequency and schedule. Meanwhile, Karl hand-crafted HTML for a prototype web site for the new journal, and other members of the editorial board made many comments on it. We then embarked on a collaboration to build a production program to generate a web site similar to the prototype. The initial issue of *The PracTEX Journal* was published (posted at `http://tug.org/pracjourn`, which remains the journal's home) on January 15, 2005. This web site includes a table of contents for each issue with links to the PDFs of the papers, and author, title, and BIBTEX lists covering all issues.

The relevance of the *PracTEX Journal* effort to *TUGboat* online is that it forced us to think extensively about how a program to generate the web site for an online journal would work. Dave wrote a lot of code, and we noted some things to be done better or differently if another journal web site generation program was ever written.

3.2 Contents, ordered by difficulty

In early 2005 Dave and Karl began discussing the possibility of creating an additional table of contents for each issue of *TUGboat* that would be organized by level of difficulty. Karl took the issue to Barbara Beeton and the other members of the *TUGboat* production group, who supported the idea. *TUGboat* Volume 25 (2005), No. 2 (*TUGboat* issue 81) was the first issue that included this additional table of contents: see `http://tug.org/TUGboat/tb25-2/cover3.pdf`. The design was discussed extensively in advance, and has had only minor refinements since that first appearance.

From the beginning, we knew we would want this information to be available online (in HTML) as well as in print (via PDF). So, we devised a format for the source file that both could be compiled by TEX to produce PDF, and would also be plausible to parse with a Perl program to generate HTML for the online contents page. We informally refer to these dual-purpose source files as "capsule files".

The capsule files begin with some prologue definitions relevant only to TEX, followed by the main material: a sequence of `\capsule` commands, each one a capsule summary of an item in the table of contents. The `\capsule` entries are in the order they would appear in the printed level-of-difficulty table of contents (since Perl can do reordering more easily than TEX). The `\capsule` command evolved over time, eventually having nine arguments:

1. The difficulty rating — follows `\capsule` on first line (example given below).
2. The *TUGboat* category, possibly with a "subargument" `replace` or `add`, as described below. Arguments 2–7 are always alone on their lines.
3. The author(s).
4. The title.
5. A one-line description for the item.
6. The page number, possibly with an `offset` subargument.
7. The URL of the item's PDF.
8. Optional subtitles. These can be multiple lines.
9. Optional additional text for the HTML. This can also be multiple lines.

(Some of the subtleties of the `\capsule` command's interpretation will be described in the next section.)

For ease of parsing by the Perl program, the arguments to `\capsule` are mostly on a single line (the exceptions are the last two, which can be multiple lines, as noted above). Here's an example from the present issue:

```
\capsule{Introductory}
  {Resources}
  {Jim Hef{}feron}
  {Which way to the forum?}
  {review of the major online help forums}
  {\getfirstpage{heff}}
  {/TUGboat/!TBIDENT!heff.pdf}
  {}
  {}
```

4 Writing a program to generate *TUGboat* contents pages

Even given the prior experience with *The PracTEX Journal* (described in section 3.1), the program to generate the *TUGboat* contents pages and lists of authors, titles, and categories/keywords evolved over a fairly long period. As we handled more of *TUGboat*'s 100 issues, we continually learned of more variations with which we had to cope.

4.1 The series of steps

Our memory of the evolution is as follows.

a. Dave first cobbled together a program to convert the capsule files (which only existed from issue 81 on) into tables of contents for new issues as they came out. In this way, we began to understand the task at hand and had real examples to review as we decided what to do next.

For this program, Dave reworked parts of the *PracTEX Journal* program. In particular, for this and all additional evolutions of the project he used a pair of template routines he developed (originally to generate HTML for the *PracTEX Journal* effort), in order to avoid learning about Perl's template capabilities; see `http://walden-family.com/public/texland/perl-common-code.pdf`.

b. Then we started to process issues prior to #81. The material we had to work with was (a) the online contents pages, which existed as hand-crafted HTML (mentioned in section 2); and (b) the so-called "`.cnt` files" created by Barbara Beeton; these are TEX files, covering (at the time) the tables of contents for all but the last few years prior to issue 81.

(Aside: The `.cnt` files are also used by Nelson Beebe to produce his *TUGboat* bibliography files. Both are available on CTAN at `http://mirror.ctan.org/info/digests/tugboat`.)

c. Where only HTML files existed, Dave converted the HTML into capsule files using a combination of Perl programming, editor macros, and manual editing. At this point, the URLs for the PDFs of the individual papers already existed in the HTML for transfer to the capsule files. (When posting the scans, Karl had updated the HTML contents files with these links to PDFs for individual items.)

d. For the years which had a `.cnt` file available, Dave wrote another Perl program for conversion into capsule files. In this case, Dave then looked at the HTML files for those issues and transferred the URLs for the PDFs into the capsule files. Dave did this manually, looking at each PDF file as he went along, spotting instances of multiple short or related papers in a single PDF file such that the same URL would be the argument to more than one `\capsule` command. (This happened because sometimes Karl had combined several items into one PDF, according to his subjective idea of when splitting would be more hindrance than help to readers.) In a few instances, we split PDF files that had previously been combined.

e. At this point, we had capsule files for all issues, and some insight into the special cases that needed to be handled. Dave then did a major renovation and expansion of the main conversion program (paragraph a). In addition to generating the tables of contents for each issue, the new version of the program created author, title and keyword lists across all issues.

f. Dave reran the new version of the program on the latest issues (see paragraph a) to make sure those capsule files still converted correctly.

g. Starting with the first issue, Dave then ran the new version of the program on successive issues, fairly often discovering new situations the program had to be augmented to handle and discussing possible solutions with Karl. Some of these situations and solutions are described in the next sections.

h. Karl and Barbara reviewed the results and suggested additional refinements. Some of these just required fixes to the capsule files to the program. Others required changes to the program.

i. Finally, we felt ready to make the full set of computer-generated web pages publicly available. They are all linked from `http://tug.org/TUGboat/contents.html`.

Over time, as new *TUGboat* issues have been produced a little additional program maintenance and improvement has been required. Altogether there have been about 50 iterations of the program.

4.2 Program structure

The program is driven by several plain text files:

- A file giving translations of words with diacritical marks, etc., both from TEX into HTML for web site display and from TEX into plain text for alphabetic sorting.
- A file for unifying both (a) different versions of the same author's name, defining a single version of the name which is used for all when sorting items for the author list), and (b) different versions of the same *TUGboat* article category (e.g., 'Fonts', 'Font Forum', 'Font Design and New Fonts', and 'Fonts and Tools'), again defining a single version which is used for the category/keyword list.
- A file listing the *TUGboat* issues to be processed in this run.
- The capsule file for each issue.

Examples of all the files discussed here can be seen at `http://tug.org/TUGboat/tb32-1/tubonline`.

The program reads the first two files to prime its data structures and then begins processing the third file, one issue number at a time, which in turn involves processing the capsule file for that issue. As each capsule file is processed, the necessary information is saved for the title, author, and keyword/category lists. The HTML contents page for each individual issue is also output in turn. After all the issues have been processed, the saved information for the three types of lists is sorted appropriately, and the web pages for these lists are created.

Thus, the program is not unusual: parsing, processing, output. It is inherently a bit messy because of different combinations of situations that must be handled in different ways, for example, the different combinations of the title, author, *TUGboat* category, and PDF URL that may be present, resulting in different output formats.

The web site generation program skips over the TEX at the beginning of the file until it reaches the first of the few TEX commands it understands, for instance, `\issue{25}{2}{81}{2004}{}{}` which indicates the year, volume number and issue number within the volume, and overall issue sequence number, starting at 1 for the first issue of *TUGboat*.

4.3 Program capabilities

Some of the capabilities of the program have already been mentioned, such as its conversion of the TEX with diacritical marks for a person's name into HTML with its equivalent diacritics for display on the web pages, and also into the strictly English A–Z and a–z alphabet for sorting. Unifying different variations of author names and *TUGboat* categories has also been previously mentioned.

The program must display slightly different table of contents pages for proceedings than for non-proceedings issues. Additionally, twice (to date) something that was not a normal *TUGboat* issue was distributed by TUG such that we nevertheless wanted to handle it as if it were a *TUGboat* issue: the preprints for the 2004 proceedings (which filled the role of *TUGboat* issue #79), and the EuroTEX 2005 proceedings (which filled the role of *TUGboat* issue #85). These instances require different formatting.

Sometimes a capsule file has text to be included in the online table of contents that doesn't appear in the printed edition (argument #9 to `\capsule`, see section 3.2). This is typically used for links to videos of conference presentations and supplementary information for an article. Our program handles these commands by simply passing along the HTML to the output; the TEX macros for generating the printed cover ignore these commands.

We want the items in the online contents to be ordered according to their appearance in the issue. However, from issue 81 on, the items in the capsule file are in the correct order for generating the level-of-difficulty table of contents, so the program has to resort them by page number. The items in capsule files before issue 81 are generally in page number order, but even then, sometimes the start-of-article page numbers are insufficient. Multiple

items can appear on the same page, some page numbers have a non-numeric format, e.g., c3 for cover page 3 (the inside back cover), and other occasional special cases. Thus, an optional `\offset{.1}`, (or `{.2}`, etc.), parameter may follow a page number in the page-number argument to a `\capsule` command, and these values are used to adjust the order of things in the page-number sort. `\offset` is ignored by TEX.

Another feature eliminates a potential source of error. In the example at the end of section 3.2, the page number is not given directly: it's specified as `\getfirstpage{heff}`. Seeing that, the program looks for an auxiliary file `heff/firstpage.tex`, as our production directory is arranged. TEX itself creates that file automatically when an issue is run, thus avoiding manually entering page numbers.

The example shows one more small feature: the URL is given as `/TUGboat/!TBIDENT!heff.pdf` instead of `/TUGboat/tb32-1/tb100heff.pdf`. The curious directive `!TBIDENT!` came about after we had been creating capsule files for new issues for some time. We realized that (naturally enough) new capsule entries were usually created by copying old ones, and it was regrettably easy to neglect updating the issue number (`32-1`), the sequence number (`100`), or both. `!TBIDENT!` simply expands to that sequence; the actual values needed for the program to do the expansion are given in the `\issue` command mentioned above.

We also added some consistency checking to the program for more help in discovering typos, etc.:

- the PDF files specified in the URLs must actually exist (this assumes the program is being run on `tug.org` itself with full access to the archive of PDFs of individual articles);
- everything in the capsule file must end up in the output, i.e., no two items have identical page numbers (we manually differentiate with `\offset` when that would otherwise be the case).

5 Opportunities

Having the archives of *TUGboat* available online made it possible for people to access articles in old issues without having to find a physical copy of the issue. The attendant full lists of authors, titles, and keywords are particularly useful to researchers (ourselves included) trying to find things in the *TUGboat* archive. For example, we have used those lists in doing background research for the TUG interview series (`http://tug.org/interviews`), and in creating the book *TEX's 2^5 Anniversary*, which involved creating lists of papers from *TUGboat*. (A fuller description of how that latter book came about is at `http://tug.org/store/tug10/10paper.pdf`.)

From the outset of *TUGboat* online, issues more than a year old were made publicly available. A year after the publication of each new issue, it was posted to the *TUGboat* web site. Following this precedent, we immediately put all older issues online in full as the material became available.

Having so much more of *TUGboat* available online brought home the question of the extent to which non-members of TUG would have access. Eventually, the TUG board of directors decided to put each issue online for members to access as soon as the paper copy was published, but to wait a year before making it available to non-members. At about the same time, members were offered a membership option (with a discount from the regular cost of membership) where they were not sent paper copies of *TUGboat* and only accessed it online. About 15 percent of members choose this option.

In general it seems that having *TUGboat* online is good for both TUG members and other people interested in TEX, typography, and fonts. Having *TUGboat* online is consistent with TUG's mission:

> TUG is a not-for-profit organization by, for, and of its members, representing the interests of TeX users worldwide and providing an organization for people who are interested in typography and font design.

As for the specifics that we've presented here, having the machine readable capsule files for all issues of *TUGboat* allows easier reuse of *TUGboat* data in other, perhaps as yet unforeseen, contexts.

Acknowledgments

Barbara Beeton, Mimi Burbank, and Christina Thiele all searched their memories and archives to help us.

As described in the beginning of the article, Mimi initiated the process of getting *TUGboat* issues online (among many other *TUGboat* and TUG and TEX efforts). We'd like to dedicate this article to her memory; sadly, she passed away late last year (a full memorial appears elsewhere in this issue).

◇ Karl Berry and David Walden
`http://tug.org/TUGboat/Contents`

TEX consulting for fun and profit

Boris Veytsman

1 About this article

At TUG 2008 I made a talk about my experiences as TEX consultant. Sometimes I have been asked to write it down. While the video of that talk is available [7], it might still be worthwhile to put to paper my observations and the lessons I learned during the years of paid consultancy. Thus I decided to write down some things I discussed in 2008 — and some new thoughts and observations.

It goes without saying that everything in this article is based on my personal experience and reflects my tastes and opinions. When I mention "lessons", they are lessons for me: the reader is free to consider them either trivial or erroneous (or even both simultaneously). Also, the words *I, my, me* are used in this paper more frequently than is usual in technical or scientific literature; again this is caused by the chosen genre of personal reminiscences.

I am grateful to Karl Berry, who encouraged me to write this paper, to Dave Walden whose interview [8] rekindled my memories, to my numerous customers, who provided many valuable lessons, and to the TEX community, without whose energy and dedication my career as a consultant would be impossible.

2 First steps

I started to use TEX in the middle of the 1990s, and quickly fell in love with it. In the first years TEX was for me synonymous with LATEX; my first book about TEX was Lamport's manual [5]. I quickly appreciated the possibilities for extension and customization in the TEX system, and decided to write my own packages. I've read the Guide [1] and the first edition of the Companion [3] — the one that featured St. Bernard with a barrel on the cover. I remember spending a whole day reading the descriptions of all the LATEX packages on CTAN — it was still possible then to do this in one day.

My first packages were just this — something for my own use. I was sending out many letters (a usual pastime for a postdoc), so I wrote the envlab package for printing envelopes. A journal where my co-author and I published a paper required an alphabetic list of symbols used in equations, so I wrote the first version of the nomencl package (later its development was taken over by others). LATEX programming requires a working knowledge of TEX, so I studied *The TEXbook* [4]. I still often consult it and the great reference manual by Victor Eijkhout [2].

After several years of personal use and occasional packages published on CTAN, I got used to giving TEX advice to colleagues and students. Once I opened *TUGboat* at the page with the advertising rates for TEX consultants, and found out that they were ridiculously low: $35 for a full year of listing both in the print and web editions. Thirty five dollars is a nice sum for an experiment: even if you fail, it is not too much to lose, and even one consulting engagement would justify many years of advertisement. Thus I sent money to TEX Users Group. This was in the end of 2005. The first offer came in about six months after this.

My ideas about the consulting were rather nebulous. Probably something along the lines of interpreting cryptic TEX error messages. However, my first engagement was completely different: a publisher of scientific journals wanted to train the help desk staff to troubleshoot authors' problems with LATEX. This was a rather lucky beginning for me: I have been teaching physics on different levels from grade school to graduate school since the 1980s. It was relatively easy for me to create a course (mostly based on the great book by Oetiker *et al.* [6]) and teach it in a two-day seminar. By the way, I was pleasantly surprised by the help desk staff: they turned out to be smart people, interested in learning and asking great questions. It was a fun assignment. I spent some time in preparing the materials, but then I was able to use them several times more for other audiences: TEX training turned out to be an important part of my business.

In 2007 I got a couple more engagements. They were from publishers; I was to program a LATEX style based on their design. Actually this activity turned out to be the main part of my consulting. Again, I was lucky: from the beginning I got to know very good designers and learned to interact with them.

This does not mean that I never got to "interpreting TEX error messages": occasionally I get desperate pleas for help from people stuck with compilation of their documents. Unfortunately for me (and fortunately for them), these problems are usually easy to solve. More often than not they are subject to my "15 minutes rule": I do not charge for an occasional quick advice that takes less than a quarter hour of my time. The main drivers of my business are packages for publishers and seminars for TEX users.

3 Some lessons learned

3.1 Fun and profit

While doing anything it is useful to reflect *why* you do it, what do you want to get from the activity?

The title of this article includes the words "fun and profit"; let us discuss them.

As to the profit, I found out that TEX consulting brings more than I thought when I sent these $35 to TUG. It easily pays for travel to TUG conferences, for books about typography, fonts and typesetting etc. Even after these business expenses there is some money left. On the other hand, this consulting is definitely not something one would retire upon or quit the day job for. Of course, I never put much effort into expanding my business; the advertising budget is still dominated by the yearly *TUGboat* ad. I wanted this consulting to be a modestly paying hobby, and I got pretty much what I wanted.

The fun element is much more prominent in this business. I enjoy learning new stuff — and I learned a lot! I now program in TEX much better than I did when I wrote only for myself. I learned many odds and bits of typesetting: what do Hebrew books use for front matter page numbers, why Hans Straßer becomes Hans Strasser in small caps font, what is the difference between early Tschichold and mature Tschichold, how many degrees are in the humanist axis tilt, and many other exciting things. If you like book-related trivia, consulting is a good way to collect it. Also, it is a good way to meet interesting people and get acquainted with many different businesses. Among my customers were editors, engineers, geologists, mathematicians, philosophers, physicians, publishers, statisticians, typographers, ... — and I learned a bit from each of them.

3.2 Why TEX is not enough

While learning non-TEXnical things is a part of the fun of consulting, it is also a requirement of the job. The customers rarely have purely TEXnical problems: more often than not they want solutions for other aspects of their typesetting or even the general information processing work flow. If a consultant can help them with these, he or she becomes more valuable. For example, most customers expect you to edit their `.bst` files. They may or may not realize that the language there is completely different from TEX — you are supposed to know both. Similarly you should be ready to answer questions about graphics and graphics manipulation, typography, options for Ghostscript runs, and many other things.

Sometimes these questions become rather exotic. One of my customers wanted to put hyperlinks from the table of contents *and* the index into a PDF file. While the hyperref package, of course, would do this easily, the customer wanted to work with the PDF created by quite different tools. An evening reading PDF specifications gave me the idea how to do this. This probably was as far from TEX as one can get on this job.

3.3 People one meets

The success or failure of a consultant ultimately depends on the customers. Somehow I (almost always) met really good customers, and most of my projects were a pleasure. In this section I would like to talk about these customers.

Since I have no permission to name the names (and it would probably not be ethical for me to do so), I will talk about composite categories of the people I worked with.

I learned many things from the *typographic designers.* These are the people who write the specifications for the styles I coded. In the ideal case the designer understands the ways of TEX; unfortunately such designer is a rarity. Thus it is the job of a TEXnician to learn to understand the designers and follow their ideas. I was lucky to meet really good designers and to have a privilege to work with them.

Technical editors are the primary consumers of the code I wrote. A consultant has to learn how they work, what do they expect, their customs and conventions. The good and trusting communication with the editors is probably one of the most important things in this line of business.

Of course, another class of consumers is the *authors.* Most TEX-writing authors think that they know TEX, and some of them really do. For the author of a package they are the tireless testers and submitters of bug reports. No editor ever gives such stress test to my styles as an author wanting to use a dozen incompatible packages with it. The best author, of course, is the one who reads the manual. Unfortunately, many authors never bother to do this. I provide lifetime support for my packages; more often than not my answers to the queries include the phrase "as explained on page *NN* of the manual..." Still, it is worth remembering that we work for the authors, not the other way round.

A separate category is the *self-publishing authors*, i.e. the authors who are simultaneously editors and publishers of their works. Usually these are students preparing their theses for submission. In this case they have rather harsh deadlines and strict requirements for the output. Often they are very anxious whether they can satisfy these deadlines and requirements, so a part of the job is to reassure them that everything is going to be done in time and right.

3.4 Some tips

During the years of my consulting I learned some things which turned out to be useful outside this occupation.

When you work simultaneously on many different projects, it is necessary to keep track of the things you are doing. Therefore a version control system is indispensable. I made a habit of putting all code and documentation under version control from the beginning of the project, and regularly committing changes. This habit has saved me many times.

Another important thing is to write detailed documentation. It is difficult to keep in mind all the details of all the projects. Many times when I got an e-mail from an author or editor with the questions about a style written several years before, I appreciated the fact that everything was documented and commented. Of course, for this to work the documentation should closely correspond to the code — that is why literate programming is essential for this work. Fortunately, the standard LaTeX tools for class and package writers (see [1]) provide a simple interface for literate programming. It is useful to write the user manual first, and then implement it in the code.

Artists learn to paint by studying old masterpieces. I found that reading the code of TeX wizards is one of the most direct ways to improve my skills and understanding of TeX.

I never had any business before I started this consulting, so my business advice may seem naïve. Still, I'll offer a couple of tips.

Many customers are worried about the long-term deployment of the code. I usually offer free lifetime support and bug fixing — this helps to alleviate these worries. As mentioned earlier, I also have the "15 minutes rule": if I can answer a question in 15 minutes or less, my advice is free; perhaps that's also part of the advertising budget.

Most of the customers do not know much of TeX. They do not understand what is easy and what is difficult to do. Well, if they did, they would not need consulting. In my opinion this means a certain responsibility on behalf of a consultant: to be fair, honest and to think about the customers' interests. For example, you can get and offer to write code for something which can be done by using existing packages. It would be much better for your self-esteem and for your relationship with the customer to point out that she or he can save money by reusing the existing solutions rather than paying you.

4 Conclusions

Looking back at these years of TeX consulting I feel it was a lucky idea to send an ad to *TUGboat*. I learned much about TeX and many other related and not so related things. Some of this knowledge and skills helped me in other endeavors. And it has been a lot of fun.

References

[1] LaTeX3 Project. *LaTeX 2ε For Class and Package Writers*, 2006. `http://mirrors.ctan.org/macros/latex/doc/clsguide.pdf`.

[2] Victor Eijkhout. *TeX by Topic*. Lulu, 2007. `http://eijkhout.net/texbytopic/texbytopic.html`.

[3] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley Professional, Boston, 1993.

[4] Donald Ervin Knuth. *The TeXbook*. Computers & Typesetting A. Addison-Wesley Publishing Company, Reading, MA, 1994. Illustrations by Duane Bibby.

[5] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, Reading, MA, 2nd edition, 1994. Illustrations by Duane Bibby.

[6] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. *The Not So Short Introduction to LaTeX 2ε, Or LaTeX 2ε in 141 Minutes*, May 2008. `http://mirrors.ctan.org/info/lshort`.

[7] Boris Veytsman. Observations of a TeXnician for hire. *TUGboat*, 29(3):484, 2008. `http://www.tug.org/TUGboat/Articles/tb29-3/tb93abstracts.pdf`. The talk is available at `http://river-valley.tv/observations-of-a-texnician-for-hire/`.

[8] Dave Walden. Boris Veytsman [interview at TeX interview corner]. `http://www.tug.org/interviews/veytsman.html`, January 2011.

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

Which way to the forum?

Jim Hefferon

The TEX and LATEX community is formed, in part, through forums, online places where we get together.

Old-timers often hang out where they always have been and beginners may not know where they could go, so both groups might benefit from a brief summary of the current options.

I will focus on places where you can get a TEX question answered, sticking to those sites with a good number of posts,[1] and not including specialized lists such as TEX Live's or package-specific ones. I will also focus on English-language options, simply because that is what I know.

1 texhax

This was the first real mailing list for TEX and friends — the earliest post is from 1986. Many of those early posts are about acquiring TEX so they remind us how far we have come.

There are now about a dozen posts a day. Many of the list contributors are long-time and knowledgeable members of the community. This list's atmosphere is polite, pleasant, and helpful. Almost never does a post go without an answer.

You do not have to subscribe to the list in order to post or to read the archives. The list is not moderated, that is, there is no one who approves every post in order to, for instance, keep out spam. But posts from an address that the list has never seen before must be approved, whether from a subscriber or not; the resulting delay is usually less than a day.

This list is now hosted by TUG. Subscribing or reading the archives is easy; see the signup page.[2]

2 comp.text.tex

This is a Usenet group so it has a feel rather like texhax's. It also has been around a long time, since at least February 1990.

This forum is well enough known to have a common abbreviation, ctt. It seems to me to be the place that is most often recommended when a person new to TEX asks where to get questions answered. As with texhax, many of the names that you will see here are well-known TEX experts and there is an overlap of members between the two.

This list has two to three dozen postings a day (many are announcements from CTAN). Almost all posts get answered but in my experience posting late on a Friday afternoon is not your best strategy. It is not moderated. The atmosphere is polite and helpful, although once in a great while an exchange can get sharp.

Few people today have Usenet access so the most popular way to read the list is through Google's interface.[3] This interface makes the forum easily searchable and with the length of time it has been around you can expect that the great majority of questions have already been answered. So if you have a question then you may not even have to ask.

However, this interface has the disadvantage that some spam messages appear. Another thing about this interface that some people may consider an issue is that as a commercial site, Google puts advertisements on this page (called "Sponsored links").

3 tex.stackexchange.com

This is a new forum and it is built with a web browser in mind rather than an email reader. It is part of a family of such sites, Stack Exchange,[4] and inherits its features from the common code base.

As with the traditional forums above, someone asks a question and other people send answers. Here, though, other readers vote on the answers and the ones with the most votes move to the top of the page. Thus a person reading a list of responses gets some guidance about which posts others find most valuable. You don't have to register but if you do (it is free) and your answers are consistently rated highly then you get reputation points — web cred.

High reputation users become 'trusted' and the forum is moderated by its trusted users (other users can flag something for a moderator to examine). One unusual aspect here is that trusted users may edit the questions and answers of other users, so you can find that what you wrote has been altered.

The area for TEX and friends is called {TEX}.[5] The larger family of sites is commercial and has advertisements, although at the moment ads do not seem to appear on this TEX area.

The interface is attractive, with the amenities that a modern surfer expects. For instance, you can search for questions that have already been answered. However, this community is new so there is not the decades-long background as with texhax or ctt. But it is growing fast; the number of postings seems to me to be at least comparable to that of the older forums. There are people hanging out on this site who are quite knowledgeable and most questions get answered quickly and reliably. I find the tone here professional as well as helpful.

[1] For example, I have omitted Google's LATEX Users Group and Yahoo's TEX and LATEX group.

[2] http://lists.tug.org/texhax

[3] http://groups.google.com/group/comp.text.tex

[4] http://stackexchange.com

[5] http://tex.stackexchange.com/

4 latex-community.org

Like the prior one, this new resource is directed to a web browser. Here the forum is a part of a larger site for LaTeX users[6] that offers articles, RSS feeds, and a number of other features.

This forum[7] is grouped into subtopics. A grouping has advantages and disadvantages; for myself, I find that while lurking on an all-purpose list I often pick up bits and pieces that I find useful later, whereas if I only read what I am interested in today then I miss those. But you may feel that the grouping helps you zero in on exactly what you want.

It seems to my visitor's eyes that the great majority of questions get answered but that the total number of daily posts is not as large as for the Stack Exchange site.

The LaTeX Community has advertisements. Discussions are moderated.

I find the conversation here to be both helpful and knowledgeable.

5 latex.reddit.com

The popular social news site Reddit[8] has a forum for LaTeX.[9] (*Full disclosure:* I moderate the Reddit TeX forum, although it gets so little traffic that I won't discuss it.)

As with other modern sites, readers can vote the posted links up or down and the winners float to the top of the page. In addition, visitors can comment on the links and reply to other commentators.

If you regularly visit Reddit then you can add the LaTeX sub-reddit to your stream so that items from it appear on your front page. This makes this forum part of your daily routine.

Reddit is a commercial operation and there are advertisements. Material is moderated (users can flag something for a moderator to examine).

This forum does not get many topics posted, two or three a day, and the number of responses to topics is also small, perhaps six or eight a day. Some of the posts are links to articles about LaTeX but the majority are questions. Most of the questions get answered, but in my experience for some it may take a couple of days or they may not get answered at all.

The tone is gentle and compared to the forums above there is more a feel here of beginner-to-beginner, or perhaps of intermediate-to-beginner. One thing that is great about this forum is that many of the posts are from obviously enthusiastic members of a new generation of TeX and LaTeX users.

6 Summary

The historical forums, `texhax` and `comp.text.tex` are still going strong. They have a good number of daily posts and support active communities.

The newer options, Stack Exchange and LaTeX Community, offer modern conveniences such as a system for voting on posts and answers. Both have built a regular group of contributors and are exciting additions to the online TeX landscape.

The Reddit LaTeX subsite is like the prior two, although with fewer posts and contributors. If a person is already a frequent visitor to the main site then becoming a member of this subsite is convenient.

Thus, in total, even with the limitations I put on which forums I'd consider here, TeX users have many options. To me, this says something very good about the strength and adaptability of the community of users of TeX and friends.

◇ Jim Hefferon
Saint Michael's College
Colchester, VT USA
jhefferon (at) smcvt dot edu

[6] http://www.latex-community.org
[7] http://www.latex-community.org/forum
[8] http://www.reddit.com
[9] http://latex.reddit.com

LaTeX at Distributed Proofreaders and the electronic preservation of mathematical literature at Project Gutenberg

Andrew Hwang

Abstract

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

1 Introduction

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

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

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

- *Mathematical Recreations and Essays* by W. W. Rouse Ball,
- *Philosophiae Naturalis Principia Mathematica* by Sir Isaac Newton,
- *A Short Account of the History of Mathematics* by W. W. Rouse Ball,
- *Calculus Made Easy* by Sylvanus P. Thompson.

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

2 Techniques of ebook production

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

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

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

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

2.1 Project Gutenberg and Distributed Proofreaders

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

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

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

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

DP process overview

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

The actual path of a book through DP is a bit more involved. The distributed work is separated into "proofing" and "formatting" stages. Proofing focuses on verifying the correctness of the raw words in the text, the general dictum being "match the scan". Because most DP volunteers do not speak LaTeX, the text file at the end of the proofing rounds omits most of the mathematical markup, and is far from being machine compilable. The formatting rounds add the necessary markup, including mathematics, footnotes, and sectional divisions. The output of the formatting rounds is, with minor modifications, machine compilable once the appropriate preamble has been prepended, but is still far from a completed ebook. The remaining work on a project, generically termed "post-processing" and typically comprising about 25–35% of the total production time, is performed off-line.

2.2 Coding for longevity

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

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

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

Carefully-planned macros meet all four requirements, embodying these multiple layers of structure, both clarifying the code and simplifying the task of future maintainers who wish to convert today's LaTeX files into source files suitable for the typesetting software of 2050 and beyond. Technical details of DP's current practices are surveyed in Section 4 below.

3 The structure of DP

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

DP is primarily an English-language site. For LaTeX projects, English-language books are generally preferred, though a number of books in French and German have also been produced. The site code currently restricts source files to the Latin-1

(`iso 8859-1`) encoding, so a book's language must be representable in Latin-1. (DP-Canada and DP-Europe can handle `utf-8` with some limitations.)

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

3.1 Content providing

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

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

Selecting suitable books

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

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

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

3.2 Proofing

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

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

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

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

3.3 Formatting

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

For a LaTeX project, F1 entails a similar amount of type-in to P1. Additionally, a "formatting coordinator" (see Section 4) provides a "working preamble" for the project. Volunteers are expected to test-compile each page before marking it as "done", and to check the compiled code visually against the page scan. This amount of work makes F1 the most time-consuming round for LaTeX, about 10–20 minutes' work per page.

3.4 Post-processing

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

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

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

3.5 Access requirements

Access to various activities at DP is granted according to time on site, number of pages processed, and/or

peer review of one's work. Each DP site has its own set of certification requirements. Criteria for the DP site at `www.pgdp.net` are described here.

New volunteers are immediately granted access to P1. Access to P2 is granted once a volunteer has been registered for 21 days and has proofed at least 300 pages. Certification to work in the third round of proofing is granted by application only, upon satisfactory performance under detailed human evaluation of the volunteer's proofing work. In order to apply for P3, a volunteer must have been registered at DP for at least six weeks, and have proofed at least 150 pages in P2, and formatted at least 50 pages.

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

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

Time commitments

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

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

Documentation and LaTeX requirements

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

4 DP LaTeX coding strategies

This section discusses, in some technical detail, current practices for coding LaTeX at DP. Most of these ideas are not new, but neither do they seem widely-articulated. These strategies need not be studied except by volunteers who intend to post-process, but their rationale must be consciously and continually remembered when working at DP, where the page-at-a-time interface naturally leads a formatter to focus detrimentally on small-scale document structure.

4.1 Textual content

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

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

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

4.2 Semantic structure

A document body should contain few commands explicitly specifying how a piece of text is to be typeset. Instead, the body contains mostly mnemonic,

high-level structural information: "this is a chapter", "this is a theorem", "this is a figure", and so forth.

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

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

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

All the typographical work is centralized in macro definitions.

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

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

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

4.3 Visual presentation

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

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

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

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

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

4.4 Implementation

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

A LaTeX source file obviously depends for compilability on external packages and the LaTeX kernel itself. For the LaTeX kernel and "core" packages, the need for backward compatibility helps ensure that *user interfaces* do not change. By contrast,

kernel and package *internals* are all but guaranteed to be re-written beyond recognition on a time scale of decades.

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

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

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

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

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

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

4.5 Remarks and caveats

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

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

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

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

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

Floats and inset illustrations

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

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

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

DP-specific coding tricks

Proofers and formatters at DP commonly make inline notes regarding misspellings, visual obscurities, notational inconsistencies, even factual errors. Two simple macros, `\DPnote` and `\DPtypo`, are used to leave notes in the source file. `\DPnote` is a one-argument null macro. `\DPtypo` accepts two argu-

ments, the original text and the putative correction. Changes are trivially switched on (or off) by changing one line of preamble code. Future maintainers can easily review all such changes by searching the source file for the macro name.

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

5 The future

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

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

LaTeX-capable formatters are needed simply to distribute large amounts of work among many volunteers. What takes one formatter a month can be done by ten volunteers in a few days. Encouraging students to work at DP can both provide valuable LaTeX coding practice and serve as an introduction to document design and planning.

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

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

References

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

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

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

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

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

Introducing the PT Sans and PT Serif typefaces

Pavel Farář

Abstract

This article introduces the high quality typefaces PT Sans and PT Serif released by ParaType. They cover many languages written in Latin or Cyrillic scripts.

1 Introduction

I was looking for some time for a good free font that could be used for both the Czech and Russian languages. This is not so easy as it might seem.

Most fonts containing the Latin alphabet are unsatisfactory for the Czech language due to a single accent — a special type of caron. This accent must be seen in the context of the whole word, not just one letter. We will see more about this later.

Some fonts are usable for the Czech language and do contain a Cyrillic alphabet, but with somewhat questionable quality.

After some time I found PT Sans. It was created by Russian professionals and therefore the high quality of the Cyrillics was no surprise to me. Moreover it was also very good for the Czech language and the font was nice. PT Serif appeared later and I decided to make these fonts available for the TEX community.

2 About the project

Both these typefaces were designed for the project "Public Types of Russian Federation". This project was founded on the occasion of the anniversary of the reform of the Russian alphabet by Peter the Great in 1708–1710 and was financially supported by the Federal Agency for Press and Mass Communications. The main aim of the project is to make it possible for the people of the Russian Federation to read and write in their native languages.

The idea to create a free font for all people and nationalities of the Russian Federation has existed for a long time. And the reason was simple — the existing fonts that most people could use had some problems. The fonts distributed with operating systems did not cover all languages. Moreover, they were usually done in western countries and this resulted in unsatisfactory quality for the Cyrillic alphabet, especially when used in print, that is, at higher resolutions. There have also been some projects creating free fonts for many languages, but they had similar problems.

People behind this project wanted to avoid all the shortcomings of existing fonts and so they formulated several requirements for the new fonts:

- They should be free of charge, with clear licensing; the usage should not be restricted.
- They should support as many languages as possible.
- They should be created by a professional, native, type designer.
- They should be universal typefaces for a wide range of use.
- They should be of high quality.
- They should be financially supported by the authorities of the Russian Federation.

The fonts were created by the professional font company ParaType by Alexandra Korolkova and Olga Umpeleva under supervision of Vladimir Yefimov. The first font that appeared was PT Sans; one year later came PT Serif, and there are plans for a monospaced font. The fonts are available in TrueType format and also as web fonts.

The work on these fonts took many months and many people helped with this project. Some institutions were asked for their opinions and needs. It was necessary to find as much information about the languages and their characters as possible — the fonts needed to contain letters that the designers had never seen before. Western designers were consulted about the Latin part of the fonts. There was much more work, and all this resulted in very nice professional fonts.

The fonts were first released under ParaType Free Font License, but the company was willing to release them also under other free licenses if needed. Therefore the fonts later appeared also under OFL. Both these licenses are very similar. The fonts can be modified and the modified version can be distributed, but not under the original name without explicit written permission from ParaType.

I have no plans to extend the fonts, I just needed to convert them from TrueType to Type 1 for the best usability in TEX and I wanted to keep the original name. So I asked for this permission and I used the fonts released under the original license.

3 Scripts of minority languages

The fonts cover all main and most minority languages that are used in the Russian Federation and also many languages of the neighbouring countries. Some of these minority languages are used by very few people and some letters used in these languages are therefore not very common. This makes these fonts uniquely valuable, but it brings some problems at the same time. The problems are mostly related to the fact that there is bad support for some letters.

The first type of problem is the standardization of these letters. Some letters are not included in

Unicode or the Adobe Glyph List. They therefore have non-standard Unicode values and are located in the private area. This is true not only for Cyrillic, but also for some Latin letters:

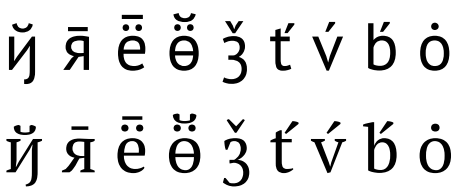


Figure 1: Some non-standard letters

The second type of problem is that some Cyrillic and Latin letters contained in the fonts are not supported by the font encodings in TEX. This is intentional for some accented Cyrillic letters because there are too many such letters and they can be constructed with quite satisfactory results from the unaccented letter and accent. But this is not true for all letters. Some uncommon Latin letters with the acute accent give very bad results. Compare the following letters with added acute accent to the letters precomposed in the PT fonts (figure 1):

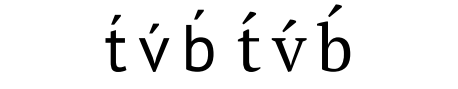


Figure 2: Composing characters with acute

There is another problem in the Cyrillic font encodings. The encodings contain several letters with descender or hook, but none with tail. The letters with tail and descender are somewhat similar, but they are used for different languages.

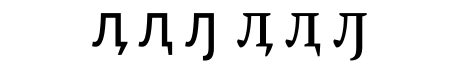


Figure 3: Cyrillic el with tail, descender and hook

For example, the Khanty language should use the Cyrillic letter el with descender, while the Itelmen language uses the same letter with tail. The encoding T2B should cover both these languages, but contains only the letter with descender.

There are several solutions to these problems, but the easiest is certainly the usage of Unicode-aware engines such as XETEX.

Another possibility is to create new font encodings covering all the missing languages — at least one Latin and one Cyrillic. This would take some time just for the PT fonts, and even more if it should be usable also for other fonts. But it would probably not be worth all the effort when there is a different and simpler solution. The future will probably belong to these new Unicode-aware engines.

4 About the fonts

Both typefaces have a fairly neutral design, with some modern humanistic features. This allows usage for many purposes and the fonts also have their own modern character.

The fonts can be used for both screen and print. PT Sans has four basic styles, two narrow styles and two caption styles. You can use PT Sans for electronic documents or the Internet, but also for printed matter of general destination. It is also well suitable for communication design like road signs or information boards.

PT Sans Regular	**PT Sans Bold**
PT Sans Italic	***PT Sans Bold Italic***
PT Sans Narrow	**PT Sans Narrow Bold**
PT Sans Caption	**PT Sans Caption Bold**

Figure 4: The styles of PT Sans

PT Serif has four basic styles and two caption styles. It is suitable for business documents and publications in various fields, including advertising and display typography.

PT Serif Regular	**PT Serif Bold**
PT Serif Italic	***PT Serif Bold Italic***
PT Serif Caption	*PT Serif Caption Italic*

Figure 5: The styles of PT Serif

Both PT Serif and PT Sans have true italics. This is not so common, especially for sans serif typefaces. There are more differences between the regular and italic shape in the Cyrillic alphabet.

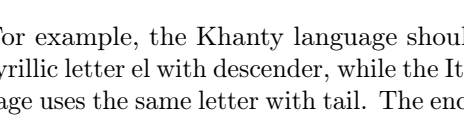


Figure 6: True italics in PT Sans

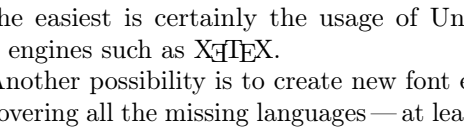


Figure 7: True italics in PT Serif

There is also another thing that is not so common in many fonts: the accents for capital and small

letters have different shapes. This has a good practical reason — it allows tighter linespacing. The accents for capital letters have smaller height than those for small letters and there are therefore fewer collisions between these accents and the descenders of letters above them.

Figure 8: Different shapes of accents

Although the families are harmonized and can be used together, PT Sans is not just PT Serif without serifs. For example the letter *g* is open in PT Sans and closed in PT Serif. The open letterform is suitable for display purposes, the closed form is good for running text.

Figure 9: Different shapes of letter g

PT Serif Caption is a font for small sizes. It has the usual changes that improve the readability of small sizes, but the designers did not stop there. They also changed some details like serifs or accents. You could overlook it for small sizes but when you use it at bigger sizes, the differences are quite obvious and you get a font with its own new character.

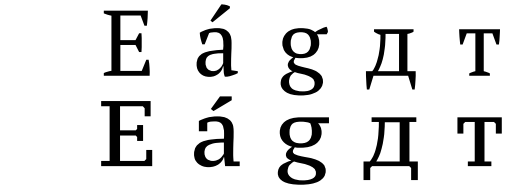


Figure 10: PT Serif and enlarged PT Serif Caption

Although the fonts were designed for the Russian Federation, their coverage of the encoding T1 is very good; only a few characters such as Ŋ and ŋ are missing. Therefore the fonts can be used for most Latin-based European languages. On the other hand, not all Cyrillic based languages are supported and the Cyrillic encodings in TEX are not fully covered. Again, just few characters are missing.

The fonts have over 700 characters, but they do not have everything. They have only ligatures fi and fl and there is only the Cyrillic em-dash that is somewhat shorter than the English one.

5 About the caron

I would like to say also some words about one usually misunderstood accent. This is the special type of caron used in small letters with ascenders. The usual type of caron could not be in the same height as it is in the letters without ascenders. Therefore a special accent is used that looks more like an apostrophe than like the usual caron.

Figure 11: Different shapes of caron

It is important to realize that this is *not* an apostrophe. It is usually more subtle and the most important difference is that words containing this accent should be compact, whereas apostrophe quite clearly separates the letters on the left from those on the right.

Figure 12: Caron and apostrophe in PT Sans

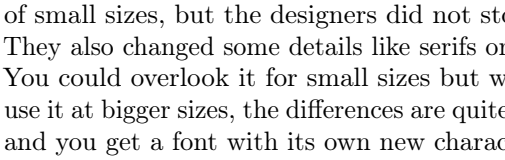


Figure 13: Caron and apostrophe in PT Serif

It looks very easy and it certainly is, but nevertheless most fonts are bad in this respect — and it is not just a TEX-related problem.

See also the Slovak language sample in section B.1 where you can see the letter *l* with caron.

6 Summary

The described typefaces have some properties that can make them a very useful part of TEX distributions:

- They are of professional quality.
- They are universal typefaces and can be used for many purposes.
- They cover many western and central European languages and they can hardly be surpassed for coverage of the languages of the Russian Federation.

The fonts are ideal for multilingual texts where you need consistent appearance for all languages. The coverage of many languages makes the fonts somewhat similar to projects like TEX Gyre, but they also contain Cyrillic letters.

A PT Sans Samples

A.1 English

All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.

All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.

A.2 Serbian

Сва људска бића рађају се слободна и једнака у достојанству и правима. Она су обдарена разумом и свешћу и треба једни према другима да поступају у духу братства.

Сва људска бића рађају се слободна и једнака у достојанству и правима. Она су обдарена разумом и свешћу и треба једни према другима да поступају у духу братства.

A.3 French

Tous les êtres humains naissent libres et égaux en dignité et en droits. Ils sont doués de raison et de conscience et doivent agir les uns envers les autres dans un esprit de fraternité.

A.4 Spanish

Todos los seres humanos nacen libres e iguales en dignidad y derechos y, dotados como están de razón y conciencia, deben comportarse fraternalmente los unos con los otros.

A.5 Czech

Všichni lidé se rodí svobodní a sobě rovní co do důstojnosti a práv. Jsou nadáni rozumem a svědomím a mají spolu jednat v duchu bratrství.

A.6 Ukrainian

Всі люди народжуються вільними і рівними у своїй гідності та правах. Вони наділені розумом і совістю і повинні діяти у відношенні один до одного в дусі братерства.

B PT Serif Samples

B.1 Slovak

Všetci ľudia sa rodia slobodní a sebe rovní, čo sa týka ich dôstojnosti a práv. Sú obdarení rozumom a svedomím a majú navzájom jednať v bratskom duchu.

Všetci ľudia sa rodia slobodní a sebe rovní, čo sa týka ich dôstojnosti a práv. Sú obdarení rozumom a svedomím a majú navzájom jednať v bratskom duchu.

B.2 Russian

Все люди рождаются свободными и равными в своем достоинстве и правах. Они наделены разумом и совестью и должны поступать в отношении друг друга в духе братства.

Все люди рождаются свободными и равными в своем достоинстве и правах. Они наделены разумом и совестью и должны поступать в отношении друг друга в духе братства.

B.3 German

Alle Menschen sind frei und gleich an Würde und Rechten geboren. Sie sind mit Vernunft und Gewissen begabt und sollen einander im Geist der Brüderlichkeit begegnen.

B.4 Danish

Alle mennesker er født frie og lige i værdighed og rettigheder. De er udstyret med fornuft og samvittighed, og de bør handle mod hverandre i en broderskabets ånd.

B.5 Polish

Wszyscy ludzie rodzą się wolni i równi pod względem swej godności i swych praw. Są oni obdarzeni rozumem i sumieniem i powinni postępować wobec innych w duchu braterstwa.

B.6 Hungarian

Minden emberi lény szabadon születik és egyenlő méltósága és joga van. Az emberek, ésszel és lelkiismerettel bírván, egymással szemben testvéri szellemben kell hogy viseltessenek.

B.7 Abkhaz

Дарбанзаалак ауаҩы дшоуп ихы дақәиҭны. Ауаа зегь зинлеи патулеи еиҟароуп. Урҭ ирымоуп ахшыҩи аламыси, дара дарагь аешьеи аешьеи реиҧш еизыҟазароуп.

References

[1] Omniglot. `http://www.omniglot.com`.

[2] ParaType. English: `http://www.paratype.com`; Russian: `http://www.paratype.ru`.

[3] Unicode Consortium. `http://unicode.org`.

◇ Pavel Farář
Prague, Czech Republic
`pavel dot farar (at) centrum dot cz`

Handling math: A retrospective

Hans Hagen

In this article I will reflect on how the plain TEX approach to math fonts influenced the way math has been dealt with in ConTEXt MkII and why (and how) we diverge from it in MkIV, now that LuaTEX and OpenType math have come around.

When you start using TEX, you cannot help but notice that math plays an important role in this system. As soon as you dive into the code you will see that there is a concept of families that is closely related to math typesetting. A family is a set of three sizes: text, script and scriptscript.

$$a^{b^c} = \frac{d}{e}$$

The smaller sizes are used in superscripts and subscripts and in more complex formulas where information is put on top of each other.

It is no secret that the latest math font technology is not driven by the TEX community but by Microsoft. They have taken a good look at TEX and extended the OpenType font model with the information that is needed to do things similar to TEX and beyond. It is a firm proof of TEX's abilities that after some 30 years it is still seen as the benchmark for math typesetting. One can only speculate what Don Knuth would have come up with if today's desktop hardware and printing technology had been available in those days.

As a reference implementation of a font, Microsoft provides Cambria Math. In the specification the three sizes are there too: a font can provide specifically designed script and scriptscript variants for text glyphs where that is relevant. Control is exercised with the `ssty` feature.

Another inheritance from TEX and its fonts is the fact that larger symbols can be made out of snippets and these snippets are available as glyphs in the font, so no special additional (extension) fonts are needed to get for instance really large parentheses. The information of when to move up one step in size (given that there is a larger shape available) or when and how to construct larger symbols out of snippets is there as well. Placement of accents is made easy by information in the font and there are a whole lot of parameters that control the typesetting process. Of course you still need machinery comparable to TEX's math subsystem but Microsoft Word has such capabilities.

I'm not going to discuss the nasty details of providing math support in TEX, but rather pay some attention to an (at least for me) interesting side effect of TEX's math machinery. There are excellent articles by Bogusław Jackowski and Ulrik Vieth about how TEX constructs math and of course Knuth's publications are the ultimate source of information as well.

Even if you only glance at the implementation of traditional TEX font support, the previously mentioned families are quite evident. You can have 16 of them but 4 already have a special role: the upright roman font, math italic, math symbol and math extension. These give us access to some 1000 glyphs in theory, but when TEX showed up it was mostly a 7-bit engine and input of text was often also 7-bit based, so in practice many fewer shapes are available, and subtracting the snippets that make up the large symbols brings down the number again.

Now, say that in a formula you want to have a bold character. This character is definitely not in the 4 mentioned families. Instead you enable another one, one that is linked to a bold font. And, of course there is also a family for bold italic, slanted, bold slanted, monospaced, maybe smallcaps, sans serif, etc. To complicate things even more, there are quite a few symbols that are not covered in the foursome so we need another 2 or 3 families just for those. And yes, bold math symbols will demand even more families.

$$a + \mathbf{b} + \boldsymbol{c} = \mathtt{d} + \mathsf{e} + \mathcal{F}$$

Try to imagine what this means for implementing a font system. When (in for instance ConTEXt) you choose a specific body font at a certain size, you not only switch the regular text fonts, you also initialize math. When dealing with text and a font switch there, it is no big deal to delay font loading and initialization till you really need the font. But for math it is different. In order to set up the math subsystem, the families need to be known and set up and as each one can have three members you can imagine that you easily initialize some 30 to 40 fonts. And, when you use several math setups in a document, switching between them involves at least some re-initialization of those families.

When Taco Hoekwater and I were discussing LuaTEX and especially what was needed for math, it was sort of natural to extend the number of families to 256. After all, years of traditional usage had demonstrated that it was pretty hard to come up with math font support where you could freely mix a whole regular and a whole bold set of characters simply because you ran out of families. This is a side effect of math processing happening in several passes: you can change a family definition within a formula, but as TEX remembers only the family

number, a later definition overloads a previous one. The previous example in a traditional TEX approach can result in:

```
a + \fam7 b + \fam8 c = \fam9 d + \fam10 e
  + \fam11 F
```

Here the a comes from the family that reflects math italic (most likely family 1) and + and = can come from whatever family is told to provide them (this is driven by their math code properties). As family numbers are stored in the identification pass, and in the typesetting pass resolve to real fonts you can imagine that overloading a family in the middle of a definition is not an option: it's the number that gets stored and not what it is bound to. As it is unlikely that we actually use more than 16 families we could have come up with a pool approach where families are initialized on demand but that does not work too well with grouping (or at least it complicates matters).

So, when I started thinking of rewriting the math font support for ConTEXt MkIV, I still had this nicely increased upper limit in mind, if only because I was still thinking of support for the traditional TEX fonts. However, I soon realized that it made no sense at all to stick to that approach: OpenType math was on its way and in the meantime we had started the math font project. But given that this would easily take some five years to finish, an intermediate solution was needed. As we can make virtual fonts in LuaTEX, I decided to go that route and for several years already it has worked quite well. For the moment the traditional TEX math fonts (Computer Modern, px, tx, Lucida, etc) are virtualized into a pseudo-OpenType font that follows the Unicode math standard. So instead of needing more families, in ConTEXt we could do with less. In fact, we can do with only two: one for regular and one for bold, although, thinking of it, there is nothing that prevents us from mixing different font designs (or preferences) in one formula but even then a mere four families would still be fine.

To summarize this, in ConTEXt MkIV the previous example now becomes:

```
U+1D44E + U+1D41B + 0x1D484 = U+1D68D + U+1D5BE
        + U+02131
```

For a long time I have been puzzled by the fact that one needs so many fonts for a traditional setup. It was only after implementing the ConTEXt MkIV math subsystem that I realized that all of this was only needed in order to support alphabets, i.e. just a small subset of a font. In Unicode we have quite a few math alphabets and in ConTEXt we have ways to map a regular keyed-in (say) 'a' onto a bold or monospaced one. When writing that code I hadn't even linked the Unicode math alphabets to the family approach for traditional TEX. Not being a mathematician myself I had no real concept of systematic usage of alternative alphabets (apart from the occasional different shape for an occasional physics entity).

Just to give an idea of what Unicode defines: there are alphabets in regular (upright), bold, italic, bold italic, script, bold script, fraktur, bold fraktur, double-struck, sans-serif, sans-serif bold, sans-serif italic, sans-serif bold italic and monospace. These are regular alphabets with upper- and lowercase characters complemented by digits and occasionally Greek.

It was a few years later (somewhere near the end of 2010) that I realized that a lot of the complications in (and load on) a traditional font system were simply due to the fact that in order to get one bold character, a whole font had to be loaded in order for families to express themselves. And that in order to have several fonts being rendered, one needed lots of initialization for just a few cases. Instead of wasting one font and family for an alphabet, one could as well have combined 9 (upper and lowercase) alphabets into one font and use an offset to access them (in practice we have to handle the digits too). Of course that would have meant extending the TEX math machinery with some offset or alternative to some extensive mathcode juggling but that also has some overhead.

If you look at the plain TEX definitions for the family related matters, you can learn a few things. First of all, there are the regular four families defined:

```
\textfont0=\tenrm \scriptfont0=\sevenrm
  \scriptscriptfont0=\fiverm
\textfont1=\teni  \scriptfont1=\seveni
  \scriptscriptfont1=\fivei
\textfont2=\tensy \scriptfont2=\sevensy
  \scriptscriptfont2=\fivesy
\textfont3=\tenex \scriptfont3=\tenex
  \scriptscriptfont3=\tenex
```

Each family has three members. There are some related definitions as well:

```
\def\rm      {\fam0\tenrm}
\def\mit     {\fam1}
\def\oldstyle{\fam1\teni}
\def\cal     {\fam2}
```

So, with \rm you not only switch to a family (in math mode) but you also enable a font. The same is true for \oldstyle and this actually brings us to another interesting side effect. The fact that oldstyle numerals come from a math font has implications for the way this rendering is supported in macro packages. As naturally all development started when TEX came around, package design decisions were

driven by the basic fact that there was only one math font available. And, as a consequence most users used the Computer Modern fonts and therefore there was never a real problem in getting those oldstyle characters in your document.

However, oldstyle figures are a property of a font design (like table digits) and as such not specially related to math. And, why should one tag each number then? Of course it's good practice to tag extensively (and tagging makes switching fonts easy) but to tag each number is somewhat over the top. When more fonts (usable in TEX) became available it became more natural to use a proper oldstyle font for text and the `\oldstyle` more definitely ended up as a math command. This was not always easy to understand for users who primarily used TEX for anything but math.

Another interesting aspect is that with OpenType fonts oldstyle figures are again an optional feature, but now at a different level. There are a few more such traditional issues: bullets often come from a math font as well (which works out ok as they have nice, not so tiny bullets). But the same is true for triangles, squares, small circles and other symbols. And, to make things worse, some come from the regular TEX math fonts, and others from additional ones, like the American Mathematical Society symbols. Again, OpenType and Unicode will change this as now these symbols are quite likely to be found in fonts as they have a larger repertoire of shapes.

From the perspective of going from MkII to MkIV it boils down to changing old mechanisms that need to handle all this (dependent on the availability of fonts) to cleaner setups. Of course, as fonts are never completely consistent, or complete for that matter, and features can be implemented incorrectly or incompletely we still end up with issues, but (at least in ConTEXt) dealing with that has been moved to runtime manipulation of the fonts themselves (as part of the so-called font goodies).

Back to the plain definitions, we now arrive at some new families:

```
\newfam\itfam \def\it{\fam\itfam\tenit}
\newfam\slfam \def\sl{\fam\slfam\tensl}
\newfam\bffam \def\bf{\fam\bffam\tenbf}
\newfam\ttfam \def\tt{\fam\ttfam\tentt}
```

The plain TEX format was never meant as a generic solution but instead was an example of a macro set and serves as a basis for styles used by Don Knuth for his books. Nevertheless, in spite of the fact that TEX was made to be extended, pretty soon it became frozen and the macros and font definitions that came with it became the benchmark. This might be the reason why Unicode now has a monospaced alphabet. Once you've added monospaced you might as well add more alphabets as for sure in some countries they have their own preferences.[1]

As with `\rm`, the related commands are meant to be used in text as well. More interesting is to see what follows now:

```
\textfont        \itfam=\tenit
\textfont        \slfam=\tensl

\textfont        \bffam=\tenbf
\scriptfont      \bffam=\sevenbf
\scriptscriptfont\bffam=\fivebf

\textfont        \ttfam=\tentt
```

Only the bold definition has all members. This means that (regular) italic, slanted, and monospaced are not actually that much math at all. You will probably only see them in text inside a math formula. From this you can deduce that contrary to what I said before, these variants were not really meant for alphabets, but for text in which case we need complete fonts. So why do I still conclude that we don't need all these families? In practice text inside math is not always done this way but with a special set of text commands. This is a consequence of the fact that when we add text, we want to be able to do so in each language with even language-specific properties supported. And, although a family switch like the above might do well for English, as soon as you want Polish (extended Latin), Cyrillic or Greek you definitely need more than a family switch, if only because encodings come into play. In that respect it is interesting that we do have a family for monospaced, but that `\Im` and `\Re` have symbolic names, although a more extensive setup can have a blackboard family switch.

By the way, the fact that TEX came with italic alongside slanted also has some implications. Normally a font design has either italic or something slanted (then called oblique). But, Computer Modern came with both, which is no surprise as there is a metadesign behind it. And therefore macro packages provide ways to deal with those variants alongside. I wonder what would have happened if this had not been the case. Nowadays there is always this regular, italic (or oblique), bold and bold italic set to deal with, and the whole set can become lighter or bolder.

In ConTEXt MkII, however, the set is larger as we also have slanted and bold slanted and even

[1] At the Dante 2011 meeting we had interesting discussions during dinner about the advantages of using Sütterlinschrift for vector algebra and the possibilities for providing it in the upcoming TEX Gyre math fonts.

smallcaps, so most definition sets have 7 definitions instead of 4. By the way, smallcaps is also special. if Computer Modern had had smallcaps for all variants, support for them in ConTEXt undoubtedly would have been kept out of the mentioned 7 but always been a new typeface definition (i.e. another fontclass for insiders). So, when something would have to be smallcaps, one would simply switch the whole lot to smallcaps (bold smallcaps, etc.). Of course this is what normally happens, at least in my setups, but nevertheless one can still find traces of this original Computer Modern-driven approach. And now we are at it: the whole font system still has the ability to use design sizes and combine different ones in sets, if only because in Computer Modern you don't have all sizes. The above definitions use ten, seven and five, but for instance for an eleven point set up you need to creatively choose the proper originals and scale them to the right family size. Nowadays only a few fonts ship with multiple design sizes, and although some can be compensated with clever hinting it is a pity that we can apply this mechanism only to the traditional TEX fonts.

Concerning the slanting we can remark that TEXies are so fond of this that they even extended the TEX engines to support slanting in the core machinery (or more precisely in the backend while the frontend then uses adapted metrics). So, slanting is available for all fonts.

This brings me to another complication in writing a math font subsystem: bold. During the development of ConTEXt MkII I was puzzled by the fact that user demands with respect to bold were so inconsistent. This is again related to the way a somewhat simple setup looks: explicitly switching to bold characters or symbols using a `\bf` (alike) switch. This works quite well in most cases, but what if you use math in a section title? Then the whole lot should be in bold and an embedded bold symbol should be heavy (i.e. more bold than bold). As a consequence (and due to limited availability of complete bold math fonts) in MkII there are several bold strategies implemented.

However, in a Unicode universe things become surprisingly easy as Unicode defines those symbols that have bold companions (whatever you want to call them, mostly math alphanumerics) so a proper math font has them already. This limited subset is often available in a font collection and font designers can stick to that subset. So, eventually we get one regular font (with some bold glyphs according to the Unicode specification) and a bold companion that has heavy variants for those regular bold shapes.

The simple fact that Unicode distinguishes regular and bold simplifies an implementation as it's easier to take that as a starting point than users who for all their goodwill see only their small domain of boldness.

It might sound like Unicode solves all our problems but this is not entirely true. For instance, the Unicode principle that no character should be there more than once has resulted in holes in the Unicode alphabets, especially Greek, blackboard, fraktur and script. As exceptions were made for non-math I see no reason why the few math characters that now put holes in an alphabet could not have been there. As with more standards, following some principles too strictly eventually results in all applications that follow the standard having to implement the same ugly exceptions explicitly. As some standards aim for longevity I wonder how many programming hours will be wasted this way.

This brings me to the conclusion that in practice 16 families are more than enough in a Unicode-aware TEX engine especially when you consider that for a specific document one can define a nice set of families, just as in plain TEX. It's simply the fact that we want to make a macro package that does it all and therefore has to provide all possible math demands into one mechanism that complicates life. And the fact that Unicode clearly demonstrates that we're only talking about alphabets has brought (at least) ConTEXt back to its basics: a relatively simple, few-family approach combined with a dedicated alphabet selection system. Of course eventually users may come up with new demands and we might again end up with a mess. After all, it's the fact that TEX gives us control that makes it so much fun.

◇ Hans Hagen
`http://pragma-ade.com`

The rules for long s

Andrew West

Abstract

This article describes the rules for the long s (ſ) in English, French, Italian, and Spanish. It appeared first online in the *BabelStone blog* in 2006, with subsequent updates.

The online PDF contains links to facsimile scans of many of the cited books, many of which are accessible via *Google Book*.

1 Introduction

In a post in my blog about the grand old trade of basket-making I included several extracts from some 18th century books, in which I preserved the *long s* (ſ) as used in the original printed texts. This got me thinking about when to use *long s* and when not. Like most readers of this blog I realised that *long s* was used initially and medially, whereas *short s* was used finally (mirroring Greek practice with regards to final lowercase *sigma* ς and non-final lowercase *sigma* σ), although there were, I thought, some exceptions. But what exactly were the rules?

Turning first to my 1785 copy of Thomas Dyche's bestselling *A Guide to the English Tongue* (first published in 1709, or 1707 according to some, and reprinted innumerable times over the century) for some help from a contemporary grammarian, I was confounded by his advice that:

> The long *ſ* muſt never be uſed at the *End* of a Word, nor immediately after the short *s*.

Well, I already knew that *long s* was never used at the end of a word, but why warn against using *long s* after *short s* when *short s* should only occur at the end of a word?

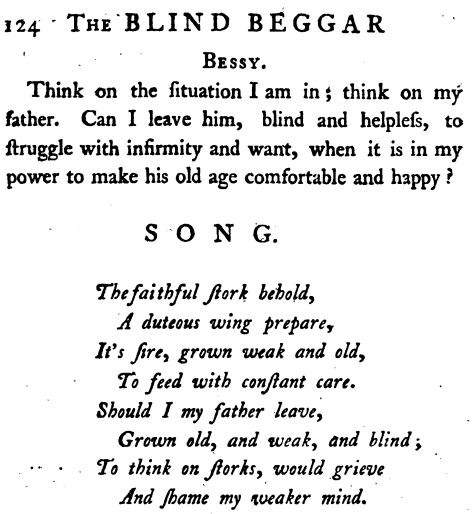
The 1756 edition of Nathan Bailey's *An Universal Etymological English Dictionary* also gives some advice on the use of *long s* (although this advice does not seem to appear in the 1737 or 1753 editions):

> A long *ſ* muſt never be placed at the end of a word, as *maintainſ*, nor a ſhort *s* in the middle of a word, as *conſpires*.

Similarly vague advice is given in James Barclay's *A Complete and Universal English Dictionary* (London, 1792):

> All the ſmall Conſonants retain their form, the long ſ and the ſhort s only excepted. The former is for the moſt part made uſe of at the beginning, and in the middle of words; and the laſt only at their terminations.

Editor's note: Werner Lemberg transformed the original blog post (with minor modifications) into this paper.



124 THE BLIND BEGGAR

BESSY.

Think on the ſituation I am in; think on my father. Can I leave him, blind and helpleſs, to ſtruggle with infirmity and want, when it is in my power to make his old age comfortable and happy?

SONG.

The faithful ſtork behold,
A duteous wing prepare,
It's ſire, grown weak and old,
To feed with conſtant care.
Should I my father leave,
Grown old, and weak, and blind;
To think on ſtorks, would grieve
And ſhame my weaker mind.

Figure 1: *The Blind Beggar of Bethnal Green* in Robert Dodsley's *Trifles* (London, 1745). In *roman* typefaces **ſ** and **f** are very similar but are easily distinguished by the horizontal bar, which goes all the way through the vertical stem of the letter 'f' but only extends to the left of the vertical stem of the *long s*; and in *italic* typefaces *long s* is even more clearly distinguished from the letter 'f' as it usually has no horizontal bar at all.

I felt sure that John Smith's compendious *Printer's Grammar* (London, 1787) would enumerate the rules for the letter 's', but I was disappointed to find that although it gives the rules for R Rotunda, the rules for *long s* are not given, save for one obscure rule (see 'Short st ligature after g' below) which does not seem to be much used in practice.

So, all in all, none of these contemporary sources are much help with the finer details of how to use *long s*. The Internet turns up a couple of useful documents: *Instructions for the proper setting of Blackletter Typefaces* discusses the rules for German Fraktur typesetting; whilst *18th Century Ligatures and Fonts* by David Manthey specifically discusses 18th century English typographic practice. According to Manthey *long s* is not used at the end of the word or before an apostrophe, before or after the letter 'f', or before the letters 'b' and 'k', although he notes that some books do use a *long s* before the letter 'k'. This is clearly not the entire story, because *long s* does commonly occur before both 'b' and 'k' in 18th century books on my bookshelves, including, for example, Thomas Dyche's *Guide to the English Tongue*.

To get the bottom of this I have enlisted the help of *Google Book Search* (see 'Note on methodology' at the end of this article) to empirically check what the usage rules for *long s* and *short s* were in printed books from the 16th through 18th centuries. It transpires that the

rules are quite complicated, with various exceptions, and vary subtly from country to country as well as over time. I have summarised below my current understanding of the rules as used in *roman* and *italic* typography in various different countries, and as I do more research I will expand the rules to cover other countries. At present I do not cover the rules for the use of *long s* in *blackletter* or *fraktur* typography, but plan to do so in the future.

2 Rules for long s in English

The following rules for the use of *long s* and *short s* are applicable to books in English, Welsh and other languages published in England, Wales, Scotland, Ireland and other English-speaking countries during the 17th and 18th centuries.

- short s is used at the end of a word (e.g. *his*, *complains*, *succeſs*)
- short s is used before an apostrophe (e.g. *clos'd*, *us'd*)
- short s is used before the letter 'f' (e.g. *ſatisfaction*, *misfortune*, *transfuſe*, *transfix*, *transfer*, *succeſsful*)
- short s is used after the letter 'f' (e.g. *offset*), although not if the word is hyphenated (e.g. *off-ſet*)
- short s is used before the letter 'b' in books published during the 17th century and the first half of the 18th century (e.g. *husband*, *Shaftsbury*), but *long s* is used in books published during the second half of the 18th century (e.g. *huſband*, *Shaftſbury*)
- short s is used before the letter 'k' in books published during the 17th century and the first half of the 18th century (e.g. *skin*, *ask*, *risk*, *masked*), but *long s* is used in books published during the second half of the 18th century (e.g. *ſkin*, *aſk*, *riſk*, *maſked*)
- Compound words with the first element ending in *double s* and the second element beginning with *s* are normally and correctly written with a dividing hyphen (e.g. *Croſs-ſtitch*,[1] *Croſs-ſtaff*[2]), but very occasionally may be written as a single word, in which case the middle letter 's' is written short (e.g. *Croſsſtitch*,[3] *croſsſtaff*[4]).
- long s is used initially and medially except for the exceptions noted above (e.g. *ſong*, *uſe*, *preſs*, *ſubſtitute*)
- long s is used before a hyphen at a line break (e.g. *neceſ-ſary*, *pleaſ-ed*), even when it would normally be a *short s* (e.g. *Shaftſ-bury* and *huſ-band* in a book where *Shaftsbury* and *husband* are normal), although exceptions do occur (e.g. *Mans-field*)
- short s is used before a hyphen in compound words with the first element ending in the letter 's' (e.g. *croſs-piece*, *croſs-examination*, *Preſs-work*, *bird's-neſt*)
- long s is maintained in abbreviations such as *ſ.* for *ſubſtantive*, and *Geneſ.* for *Geneſis* (this rule means that it is practically impossible to implement fully correct automatic contextual substitution of *long s* at the font level)

Usage in 16th and early 17th century books may be somewhat different — see 'Rules for long s in early printed books' below for details.

3 Rules for long s in French

The rules for the use of *long s* in books published in France and other French-speaking countries during the 17th and 18th centuries are much the same as those used in English typography, but with some significant differences, notably that *short s* was used before the letter 'h'.

- short s is used at the end of a word (e.g. *ils*, *hommes*)
- short s is used before an apostrophe (e.g. *s'il* and *s'eſt*)
- short s is used before the letter 'f' (e.g. *ſatisfaction*, *toutesfois*)
- short s is used before the letter 'b' (e.g. *presbyter*)
- short s is used before the letter 'h' (e.g. *déshabiller*, *déshonnête*)
- long s is used initially and medially except for the exceptions noted above (e.g. *ſans*, *eſt*, *ſubſtituer*)
- long s is normally used before a hyphen at a line break (e.g. *leſ-quels*, *paſ-fer*, *déſ-honneur*), although I have seen some books where *short s* is used (e.g. *les-quels*, *pas-fer*, *dés-honneur*)
- short s is normally used before a hyphen in compound words (e.g. *tres-bien*), although I have seen *long s* used in 16th century French books (e.g. *treſ-bien*)
- long s is maintained in abbreviations such as *Geneſ.* for *Geneſis*

4 Rules for long s in Italian

The rules for the use of *long s* in books published in Italy seem to be basically the same as those used in French typography:

- short s is used at the end of a word
- short s is used before an apostrophe (e.g. *s'informaſſero*, *fuſs'egli*)
- short s is used before an accented vowel (e.g. *paſsò*, *ricusò*, *sù*, *sì*, *così*), but not an unaccented letter (e.g. *paſſo*, *ſi*)

[1] *A Critical Enquiry into the Present State of Surgery* (1754), p. 286.

[2] *Epitome of the Art of Navigation* (1770), p. 262.

[3] *The Spectator*, No. 377 (13th May 1712), in *Harrison's British Classicks* (1786), p. 760.

[4] *The new and complete dictionary of the English language* (1795), entry for *Jacobstaff*.

- short s is used before the letter 'f' (e.g. *foddisfare*, *foddisfazione*, *trasfigurazione*, *sfogo*, *sfarzo*)
- short s is used before the letter 'b' (e.g. *sbaglio*, *sbagliato*)
- long s is used initially and medially except for the exceptions noted above
- long s is used before a hyphen in both hyphenated words and at a line break (e.g. *reſtaf-fero*)

The most interesting peculiarity of Italian practice is the use of *short s* before an accented vowel, which is a typographic feature that I have not noticed in French books.

In some Italian books I have occasionally seen *double s* before the letter 'i' written as *long s* followed by *short s* (e.g. *utiliſsima*,[5] but on the same page as *compreſſioni*, *proſſima*, etc.). And in some 16th century Italian books *double s* before the letter 'i' may be written as a *short s* followed by a *long s*. See 'Rules for long s in early printed books' below for details.

5 Rules for long s in Spanish

It has been a little more difficult to ascertain the rules for *long s* in books published in Spain as Google Book Search does not return many 18th century Spanish books (and even fewer Portuguese books), but I have tried to determine the basic rules from the following three books :

- *Estragos de la Luxuria* (Barcelona, 1736), see figure 2
- *Autos sacramentales alegoricos, y historiales del Phenix de los Poetas el Espanol* (Madrid, 1760)
- *Memorias de las reynas catholicas* (Madrid, 1770)

From these three books it appears that the rules for Spanish books are similar to those for French books, but with the important difference that (in both *roman* and *italic* type) the sequence *ſs* (not a ligature) is used before the letter 'i', whereas the sequence *ſſ* is used before all other letters (e.g. *illuſtriſsimos* but *confeſſores*):

In summary, the rules for Spanish books are:

- short s is used at the end of a word
- short s may be used before an accented vowel (e.g. *sí*, *sì*, *sé*, *sè*, *Apoſtasìa*, *Apoſtasía*, *abrasò*, *paſsò*), but not an unaccented letter (e.g. *ſi*, *ſe*, *paſſo*)
- short s is used before the letter 'f' (e.g. *transformandoſe*, *transfigura*, *ſatisfaccion*)
- short s is used before the letter 'b' (e.g. *presbytero*)
- short s is used before the letter 'h' (e.g. *deshoneſtos*, *deshoneſtidad*)
- short s is used after a *long s* and before the letter 'i' (e.g. *illuſtriſsimo*, *paſsion*, *confeſsion*, *poſsible*), but double *long s* is used before any letter other than the letter 'i' (e.g. *exceſſo*, *comiſſario*, *neceſſaria*, *paſſa*)
- long s is used initially and medially except for the exceptions noted above
- long s is used before a hyphen in both hyphenated words and at a line break, even when it would normally be a *short s* (e.g. *tranſ-formados*, *copioſiſ-fimo*)

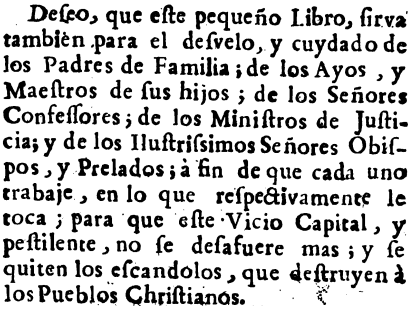

Deſeo, que eſte pequeño Libro, ſirva tambièn para el deſvelo, y cuydado de los Padres de Familia; de los Ayos, y Maeſtros de ſus hijos; de los Señores Confeſſores; de los Miniſtros de Juſticia; y de los Iluſtriſsimos Señores Obiſpos, y Prelados; à fin de que cada uno trabaje, en lo que reſpectivamente le toca; para que eſte Vicio Capital, y peſtilente, no ſe defafuere mas; y ſe quiten los eſcandolos, que deſtruyen à los Pueblos Chriſtianos.

Figure 2: *Estragos de la Luxuria* (Barcelona, 1736).

As with Italian books, Spanish books usually use a *short s* before an accented vowel, although from the three books that I have examined closely it is not quite clear what the exact rule is. For example, *Memorias de las reynas catholicas* consistently uses *short s* before an accented letter 'i' (e.g. *sí*), but consistently uses a *long s* before an accented letter 'o' (e.g. *paſſó*, *caſó*, *preciſó*, *Caſóle*); whereas *Estragos de la Luxuria* uses *short s* before both an accented letter 'i' (e.g. *sì*) and an accented letter 'o' (e.g. *abrasò*, *paſsò*).

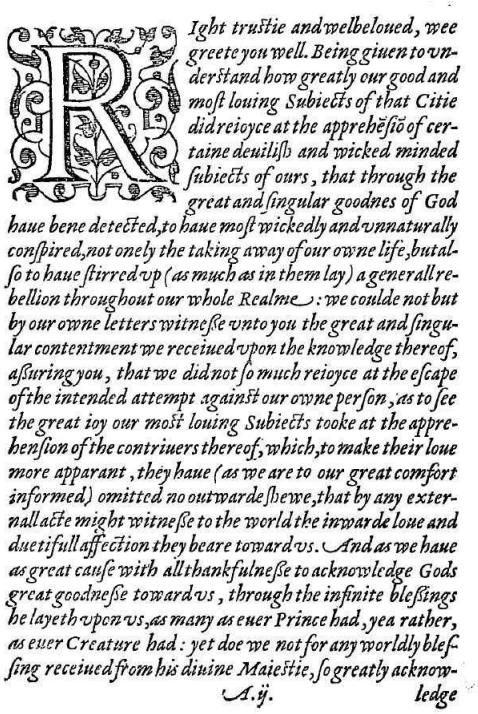
6 Rules for long s in other languages

Other languages may use rules different from those used in English and French typography. For example, my only early Dutch book, Simon Stevin's *Het Burgerlyk Leven [Vita Politica]* (Amsterdam, 1684) follows the German practice of using *short s* medially at the end of the elements of a compound word (e.g. *misverſtants*, *Rechtsgeleerden*, *wisconſtige*, *Straatsburg*, *Godsdienſten*, *misgaan*, *boosheyt*, *dusdonig* and *misbruyk*).

7 Rules for long s in early printed books

In 16th century and early 17th century books printed in *roman* or *italic* typefaces (as opposed to *blackletter*) the rules for the use of *long s* may be slightly different to those enumerated above. For example, in *italic* text it was common to use a ligature of *long s* and *short s* (ß) for double-s, whereas a double *long s* ligature was normally used in *roman* text. This can be seen in figure 3 which shows an extract from an English pamphlet published in 1586, which has the words *witneße*, *aßuring*, *thankfulneße*, *goodneße* and *bleßings*. But in that part of the same pamphlet that is set in *roman* typeface

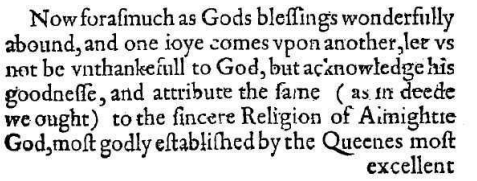
[5] *Opere di Ambrogio Bertrandi* (1787), p. 77.



Ight truſtie and welbeloued, wee greete you well. Being giuen to vnderſtand how greatly our good and moſt louing Subiects of that Citie did reioyce at the apprehẽſiõ of certaine deuiliſh and wicked minded ſubiects of ours, that through the great and ſingular goodnes of God haue bene detected, to haue moſt wickedly and vnnaturally conſpired, not onely the taking away of our owne life, but alſo to haue ſtirred vp (as much as in them lay) a generall rebellion throughout our whole Realme: we coulde not but by our owne letters witneſſe vnto you the great and ſingular contentment we receiued vpon the knowledge thereof, aſſuring you, that we did not ſo much reioyce at the eſcape of the intended attempt againſt our owne perſon, as to ſee the great ioy our moſt louing Subiects tooke at the apprehenſion of the contriuers thereof, which, to make their loue more apparant, they haue (as we are to our great comfort informed) omitted no outwarde ſhewe, that by any externall acte might witneſſe to the world the inwarde loue and duetifull affection they beare toward vs. And as we haue as great cauſe with all thankfulneſſe to acknowledge Gods great goodneſſe toward vs, through the infinite bleſſings he layeth vpon vs, as many as euer Prince had, yea rather, as euer Creature had: yet doe we not for any worldly bleſſing receiued from his diuine Maieſtie, ſo greatly acknow-

 ledge

Figure 3: *The True Copie of a Letter from the Qveenes Maiestie* (London, 1586), p. A.ii.



Now foraſmuch as Gods bleſſings wonderfully abound, and one ioye comes vpon another, let vs not be vnthankefull to God, but acknowledge his goodneſſe, and attribute the ſame (as in deede we ought) to the ſincere Religion of Almightie God, moſt godly eſtabliſhed by the Queenes moſt excellent

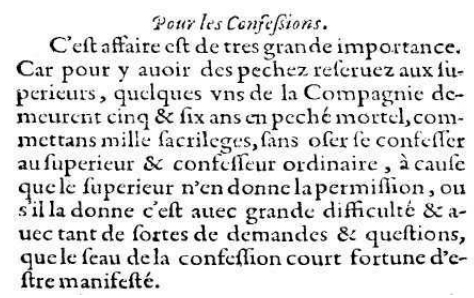
Figure 4: *The True Copie of a Letter from the Qveenes Maiestie* (London, 1586), p. A.iv.

the words 'bleſſings' and 'goodneſſe' are written with a double *long s* ligature, as shown in figure 4.

Figure 5 shows a French book published in 1615 which has *Confeßions* in *italic* type, but 'confeſſion' in *roman* type.

This ligature is still occasionally met with in a word-final position in *italic* text late into the 17th century, for example in this page from Hooke's *Micrographia* (figure 6), which has this example of the word *Addreß*, although unligatured *long s* and *short s* are used elsewhere at the end of a word (e.g. *ſmalneſs*) as well as occasionally in the middle of a word (e.g. *aſsiſted*, alongside *aſſiſtances*) in *italic* text.

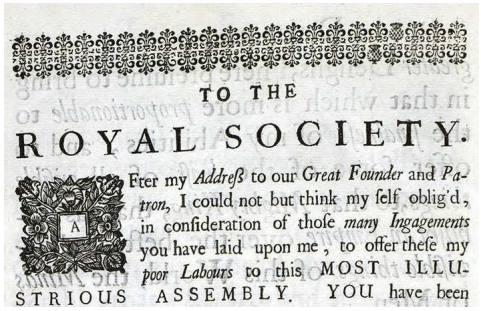
Another peculiarity that is seen in some 16th century Italian books is the use of *short s* before *long s* medially before the letter 'i', but double *long s* before any other letter; see figure 7.



Pour les Confeſſions.

C'eſt affaire eſt de tres grande importance. Car pour y auoir des pechez reſeruez aux ſuperieurs, quelques vns de la Compagnie demeurent cinq & ſix ans en peché mortel, commettans mille ſacrileges, ſans oſer ſe confeſſer au ſuperieur & confeſſeur ordinaire, à cauſe que le ſuperieur n'en donne la permiſſion, ou s'il la donne c'eſt auec grande difficulté & auec tant de ſortes de demandes & queſtions, que le ſeau de la confeſſion court fortune d'eſtre manifeſté.

Figure 5: *Advis de ce qu'il y a à réformer en la Compagnie des Jésuites* (1615), p. 13.



TO THE

ROYAL SOCIETY.

After my *Addreß* to our *Great Founder* and *Patron*, I could not but think my ſelf oblig'd, in conſideration of thoſe *many Ingagements* you have laid upon me, to offer theſe my *poor Labours* to this MOST ILLUSTRIOUS ASSEMBLY. YOU have been

Figure 6: *Micrographia* (London, 1665), p. 13.

infinitamente. Al che hauendo poi auuertito infiniti ſapienti del mondo, & conoſcendo ueramente, quanta ſia la grandezza, & l'utilità di queſta facultà diuina, inuaghiti dalla amenità, & dolcezza ſua, ſi poſero à contemplare con continuo ſtudio ogni bella, & neceſſaria parte di quella; & quella ſperialmente che narra, & inſegna la facultà marauigliosa delle piante. Del che ce ne fanno amplisſima fede Pittagora, Ariſtotele, Theophraſto, Democrito, Zoroaſtre, Xenophonte, Amphiloco, Atheneo, Hipparco, Ariſtomacho, Philiſthene, Apollodoro, Ariſtandro, Bione, Agathocle, Diodoro, Diocle, Epigene, Euagora, Praſſagora, Eraſiſtrato, Metrodoro, Hiceſio, Pamphilo, Mantia, Herophilo, Hippocrate, Crateua, Dioſcoride fra tutti gli altri celeberrimo, Galeno, Plinio, & altri infiniti antichi, i nomi de i quali per breuità trapaſſo. Imperoche coſtoro acceſi dalla giocondità, nobiltà, & grandezza di queſta utiliſsima ſcienza, dall'ardore di giouare alla poſterità uniuerſale, & dal diſio d'acquiſtarſi una fama perpetua, & immortale, non ſi ſgomentano di porre la propria uita in molti, & uarij pericoli, mentre che facendo ſmiſurati pelegrinaggi, & nauigando lunghisſimi mari, face uano ogni eſtrema fatica, & diligenza di poter conſeguire la uera, & legitima cognitione de i ſemplici, & di farſi anchor eſsi ritrouatori di molti per auanti non conoſciuti. Che ſenza dubbio ſia uero, che la ſcienza, & facultà delle piante, & parimente il ritrouarne di nuoue, oltre all'utilità, & piacer grande, che ſe ne prende l'animo, apportino lodi immortali, & perpetua fama, lo conobbero non ſolamente tutti i primi ſapienti del mondo, diligentisſimi inueſtigatori delle coſe; ma anchora molti magnanimi, & potentiſſimi Re di Corona. percioche marauigliandoſi della chiarezza del nome di

Figure 7: *I Discorsi di M. Pietro And. Matthioli* (Venice, 1563). Note the words *amplisſima, utilisſima, longhisſimi, diligentisſimi*, etc., but *potentiſſi-mi* at the end of the second to last line; cf. *neceſſaria, Praſſagora, trapaſſo.*

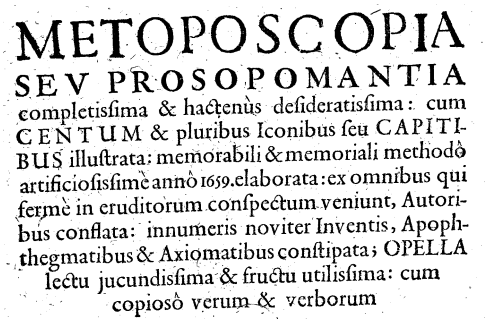

METOPOSCOPIA
SEV PROSOPOMANTIA
completisſima & hactenùs deſideratisſima: cum
CENTUM & pluribus Iconibus ſeu CAPITI-
BUS illuſtrata: memorabili & memoriali methodò
artificioſisſimè annò 1659. elaborata: ex omnibus qui
fermè in eruditorum conſpectum veniunt, Autori-
bus conflata: innumeris noviter Inventis, Apoph-
thegmatibus & Axiomatibus conſtipata; OPELLA
lectu jucundisſima & fructu utilisſima: cum
copioso verum & verborum

Figure 8: Title page to *Metoposcopia* (Leipzig, 1661). Note the words *completisſima*, *deſideratisſima*, *artificioſisſimè*, *jucundisſima*, *utilisſima*.

This typographic feature can also be seen in some later books (as shown in figure 8), though I am not yet sure how widespread it was.

8 Short s before and after f

In 17th and 18th century English and French typography the main exceptions to the rule that *short s* is not used at the start of a word or in the middle of a word is that *short s* is used next to a letter 'f' instead of the expected *long s* (so *misfortune* and *offset*, but never *miſſortune* or *offſet*). The reason for this must be related to the fact that the two letters *ſ* and *f* are extremely similar, although as the combination of the two letters does not cause any more confusion to the reader than any other combination of *long s* and another letter (the combinations *ſl* and *fl* are far more confusable) it does not really explain why *long s* should be avoided before or after a letter 'f', other than perhaps for aesthetic reasons. In all probability the rule is inherited from *blackletter* usage, as is evidenced by the 1604 pamphlet shown in figure 9 about a mermaid that was sighted in Wales, which has **fatisfaction**.

Whatever the reasons, this is an absolute rule, and Google Book Search only finds a handful of exceptions from the 17th and 18th century, most probably typographical errors (or in the case of the Swedish-English dictionary due to unfamiliarity with English rules):

- *miffortune* in *Anglorum Speculum* (London, 1684) [but *misfortune* elsewhere]
- *fatiffie* and *fatiffied* in *The Decisions of the Lords of Council and Session* (Edinburgh, 1698)
- *miffortune* in *The annals of the Church* (London, 1712) [but *misfortune* elsewhere]
- *miffortune* in *An Historical Essay Upon the Loyalty of Presbyterians* (1713) [but *misfortune* elsewhere]
- *fatiffaction* in *An Enquiry Into the Time of the Coming of the Messiah* (London, 1751) [but on the same page as *fatisfied*]
- *miffortune* in *Svenskt och engelskt lexicon* (1788)

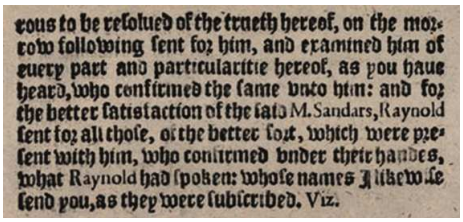

eous to be reſolued of the trueth hereof, on the mor-
row following ſent for him, and examined him of
euery part and particularitie hereof, as you haue
heard, who confirmed the ſame vnto him: and for
the better ſatisfaction of the ſaid M. Sandars, Raynold
ſent for all thoſe, of the better ſort, which were pre-
ſent with him, who confirmed vnder their handes,
what Raynold had ſpoken: whoſe names I like wiſe
ſend you, as they were ſubſcribed. Viz.

Figure 9: 1604 pamphlet about a mermaid.

Similarly, Google Book Search finds 628 French books published between 1700 and 1799 with *fatisfaction* but only two books with *fatiffaction*.

9 Short s before b and k

As a general rule English books published in the 17th century and the first half of the 18th century have a *short s* before the letters 'b' and 'k' (so *husband* and *ask*), whereas books published during the second half of the 18th century have a *long s* (so *huſband* and *aſk*). This is not a hard and fast rule, as it is possible to find examples of books from the 17th and early 18th century that show *huſband* and *aſk*, but they are few and far between. For example, whereas Google Book Search finds 138 books published between 1600 and 1720 with *husband*, Google Book Search only finds nine *genuine* books from this period that have *huſband* (excluding false positives and hyphenated *huf-band*), and in almost all cases *huſband* is not used exclusively :

- *The Dutch Courtezan* (London, 1605) [mostly *husband* but a couple of instances of *huſband*]
- *The breast-plate of faith and love* (London, 1651) [mostly *husband* but one instance of *huſband*]
- *Tryon's Letters, Domestick and Foreign, to Several Persons of Quality* (London, 1700)
- *The Present State of Trinity College in Cambridge* (London, 1710) [mostly *husband* but one instance of *huſband*]
- *Dialogue between Timothy and Philatheus* (London, 1711) [one instance each of *husband* and *huf-band*]
- *The Universal Library; Or, Compleat Summary of Science* (1712) [two instances of *huſband*]
- *The Works of Petronius Arbiter* (London, 1714) [mixture of both *husband* and *huſband*]
- *Letters Writ by a Turkish Spy* (London, 1718) [mostly *husband* but a couple of instances of *huf-band*]
- *An Historical Essay Concerning Witchcraft* (London, 1718) [mostly *husband* but one instance of *huſband*]

Likewise, it is possible to find books from the late 18th century that use *long s* but show *husband* and *ask*,

Date	husband	hufband	ask	afk	presbyter(e)	prefbyter(e)
1640-1649	6	0	4	0	1	0
1650-1659	12	3	12	1	1	0
1660-1669	4	1	10	2	1	0
1670-1679	5	1	10	1	0	1
1680-1689	12	0	22	0	3	0
1690-1699	4	0	5	0	2	0
1700-1709	39	1	54	0	3	0
1710-1719	42	8	74	7	8	2
1720-1729	49	8	78	11	7	1
1730-1739	53	25	87	36	13	1
1740-1749	44	46	50	66	11	0
1750-1759	37	168	43	201	12	2
1760-1769	36	286	30	307	11	2
1770-1779	22	320	21	342	26	5
1780-1789	27	337	21	368	37	1
1790-1799	65	404	71	464	21	1

Table 1: Change of spellings.

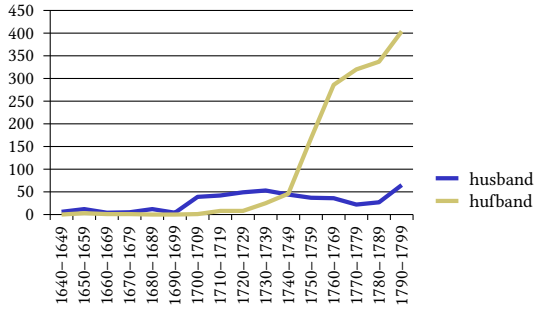


Figure 10: *husband* vs. *hufband* 1640–1799.

but these are relatively few in number. For example, whereas Google Book Search finds 444 books published between 1760 and 1780 that have *hufband*, it only finds 60 that have *husband* (excluding false positives on *HUSBAND*).

The results of Google Book Search searches on the two spellings of *husband* and *ask* (as well as *presbyter(e)* in French books) from 1640 to 1799 are shown in table 1 in ten-year segments (matches for *HUSBAND* and *ASK* have been discounted, but otherwise figures have not been adjusted for false positives such as *huf-band*).

The change in the usage of *short s* to *long s* before 'b' and 'k' appears even more dramatic if these figures are plotted on a graph, as displayed in figures 10 and 11.

But for French books, no change in rule occurred in the middle of the century, and *short s* continued to be used in front of the letter 'b' throughout the 18th century, as can be seen from the distribution of the words *presbyter(e)* and *prefbyter(e)* in figure 12.

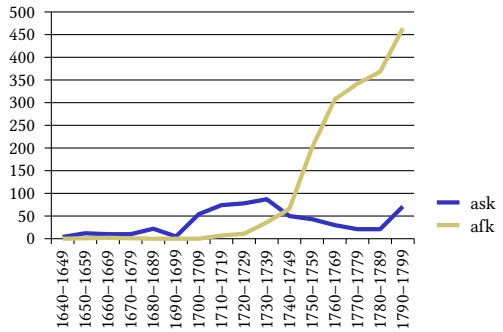


Figure 11: *ask* vs. *afk* 1640–1799.

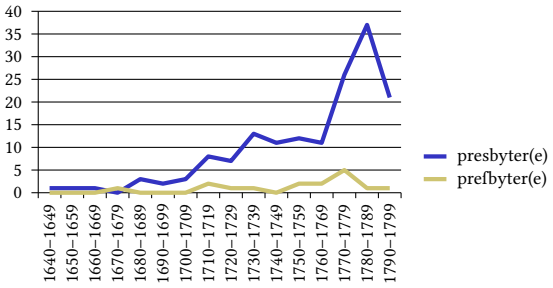


Figure 12: *presbyter(e)* vs. *prefbyter(e)* 1700–1799.

So why then did the change in rule for 's' before 'b' and 'k' happen in England during the 1740s and 1750s? According to John Smith's *Printer's Grammar*, p. 45, the Dutch type that was most commonly used in England before the advent of the home-grown typefaces of William Caslon did not have 'ſb' or 'ſk' ligatures, and that it was Caslon who first cast 'ſb' and 'ſk' ligatures. So with the growth in popularity of Caslon's typefaces ligatured 'ſb' and 'ſk' took the place of 'sb' and 'sk' — but further research is required to confirm to this hypothesis.

As to why this rule (as well as the French rule of *short s* before 'h') developed in the first place, I suspect that it goes back to *blackletter* usage, but that is something for future investigation (all I can say at present is that Caxton's Chaucer (1476, 1483) seems to use *long s* before the letters 'f', 'b' and 'k'). It is perhaps significant that the letters 'b', 'k' and 'h' all have the same initial vertical stroke, but quite what the significance of this is I am not sure.

10 Short s before h

French and English typographic practice differs in one important respect: French (and also Spanish) typography uses a *short s* before the letter 'h', whereas English typography uses a *long s*.

For example, Google Book Search finds 86 books with *déshabiller* or its various grammatical forms (*déshabillé*, *déshabillée*, *déshabille*, *déshabilles*, *déshabillez*

or *déshabillent*) during the period 1700–1799, but only a single book that uses *long s*: *déſhabillé* occurs three times in *Appel a l'impartiale postérité, par la citoyenne Roland* (Paris, 1795).

On the other hand, for the period 1640–1799 Google Books finds 54 books with *dishonour* and 196 books with *diſhonour*, but closer inspection shows that almost every single example of *dishonour* in pre-1790 books is in fact *diſhonour* or *DISHONOUR* in the actual text. Similar results were obtained when comparing the occurrences of *worship* and *worſhip*. Thus it seems that *short s* was not used before the letter 'h' in English typography.

11 Short st ligature after g

According to John Smith's *The Printer's Grammar*, first published in 1755, there is a particular rule for *italic* text only: that a short st-ligature is used after the letter 'g' in place of a long st-ligature (p. 23–24):

> In the mean time , and as I have before declared ; Italic diſcovers a particular delicacy , and ſhews a mathematical judgement in the Letter-cutter , to keep the Slopings of that tender-faced Letter within ſuch degrees as are required for each Body , and as do not detriment its individuals . But this precaution is not always uſed ; for we may obſerve that in ſome Italics the lower-caſe *g* will not admit another *g* to ſtand after it , without putting a Hair-ſpace between them , to prevent their preſſing againſt each other : neither will it give way to *ſ* and the ligature *ſt* ; and therefore a round *st* is caſt to ſome Italic Founts , to be uſed after the letter *g* ; but where the round *st* is wanting an *st* in two pieces might be uſed without diſcredit to the work , rather than to ſuffer the long *ſt* to cauſe a gap between the *g* and the ſaid ligature .

However, I have thus far been unable to find any examples of this rule in practice. For example, Google Book Search finds several examples of *Kingſton* in *italic* type, but no examples of *Kingston* in books that use a *long s*:

- *An Universal, Historical, Geographical, Chronological and Poetical Dictionary* (London, 1703)
- *Athenæ Britannicæ, or, A Critical History of the Oxford and Cambrige Writers and Writings* (London, 1716), p. 322
- *The History of England* (London, 1722), p. 78
- *Scanderbeg: Or, Love and Liberty* (London, 1747), p. 92
- *An Introduction to the Italian Language* (London, 1778), p. 109
- *A Collection of Treaties* (London, 1790), p. 288

12 The demise of the long s

Long s was used in the vast majority of books published in English during the 17th and 18th centuries, but suddenly and dramatically falls out of fashion at the end of the 18th century, reflecting the widespread adoption of new, modern typefaces based on those developed by Bodoni and Didot during the 1790s. In England this movement was spearheaded by the printer William Bulmer, who set the benchmark for the new typographical style with his 1791 edition of *The Dramatic Works of Shakspeare*, printed using a typeface cut by William Martin. The *ſ*-free typeface used by Bulmer can be seen in the Advertisement to his 1795 edition of *Poems by Goldsmith and Parnell* (figure 13).

ADVERTISEMENT

To raise the Art of Printing in this country from the neglected state in which it had long been suffered to continue, and to remove the opprobrium which had but too justly been attached to the late productions of the English press, much has been done within the last few years; and the warm emulation which has discovered itself amongst the Printers of the present day, as well in the remote parts of the kingdom as in the metropolis, has been highly patronized by the public in general. The present volume, in addition to the SHAKSPEARE, the MILTON, and many other valuable works of elegance, which have already been given to the world, through the medium of the Shakspeare Press, are particularly meant to combine the various

Figure 13: William Martin's *ſ*-free typeface.

Although throughout most of the 1790s the vast majority of English books continued to use *long s*, during the last two or three years of the century books printed using modern typefaces started to become widespread, and in 1801 *short s* books overtook *long s* books. The rise of *short s* and decline of *long s*, as measured by the occurrences of the word *those* compared with *thoſe* in Google Book Search, is charted in table 2 and figure 14.

The death knell for *long s* was finally sounded on September 10th 1803 when, with no announcement or any of the fuss that accompanied the typographic reform of October 3rd 1932 (see the articles in the issues of Sept. 26th and 27th 1932), *The Times* newspaper quietly switched to a modern typeface with no *long s* or old-fashioned ligatures, as shown in figure 16 (this was one of several reforms instituted by John Walter the Second, who became joint proprietor and exclusive manager of *The Times* at the beginning of 1803).

Date	those	thofe	il est	il eft
1780	2	140	1	126
1781	0	114	3	170
1782	0	112	4	161
1783	0	98	1	139
1784	3	131	8	230
1785	3	85	3	185
1786	5	132	20	180
1787	4	167	42	182
1788	4	174	59	194
1789	2	131	24	155
1790	7	191	58	124
1791	3	141	55	83
1792	9	192	80	130
1793	38	206	16	14
1794	46	292	8	14
1795	16	251	18	8
1796	44	199	51	16
1797	62	165	88	15
1798	57	152	97	12
1799	105	151	48	7
1800	108	224	86	6
1801	198	181	134	17
1802	210	202	154	4
1803	240	164	183	11
1804	336	118	131	8
1805	294	151	143	6
1806	328	130	173	9
1807	361	136	123	8
1808	418	54	166	2
1809	406	74	162	3
1810	492	66	223	4

Table 2: *those* vs. *thofe* and *il est* vs. *il eft*.

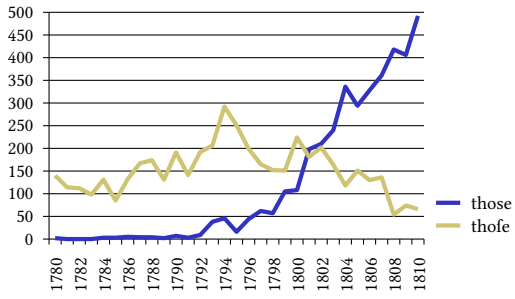


Figure 14: *those* vs. *thofe* 1780–1810.

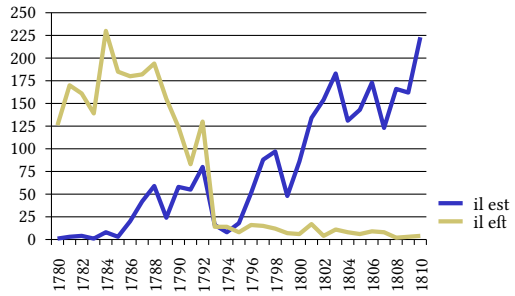


Figure 15: *il est* vs. *il eft* 1780–1810.

By the second half of the 19th century *long s* had entirely died out, except for the occasional deliberate antiquarian usage (for example, my 1894 edition of *Coridon's Song and Other Verses* uses *long s* exclusively in a medial position, with *short s* in both initial and final positions).

As might be expected, the demise of *long s* in France seems to have occurred a little earlier than in England. Based on the following Google Book Search data for *il est* and *il eſt*, it seems that *short s* started to gain popularity from the mid 1780s, and *long s* had been almost completely displaced by 1793, as shown in table 2 and figure 15 (many of the post-1792 examples of *long s* are from books published outside France).

13 Note on methodology

The statistics given here are based on the results returned from searches of Google Book Search (filtering on the appropriate language and 'Full view only'), which allows me to distinguish between words with *long s* and words with *short s* only because the OCR software used by Google Book Search normally recognises *long s* as the letter 'f', and so, for example, I can find instances of *huſband* by searching for 'hufband'. However, for a number of reasons the results obtained are not 100% accurate.

Firstly, the search engine does not allow case-sensitive searches, so whereas searching for 'hufband' only matches instances of *huſband*, searching for 'husband' matches instances of both *husband* and *HUSBAND*, which skews the results in favour of *short s*.

Secondly, hyphenated words at a line break may match with the corresponding unhyphenated word, so searching for 'hufband' may match instances of *huſband*, which is not relevant as *long s* is expected before a hyphen (Google Book Search shows 583 matches for *huf-band*, but only 3 for *hus-band* for the period 1700–1799).

Thirdly, *long s* is sometimes recognised by the OCR software as a *short s*, especially when typeset in italics.

Fourthly, the publication date given by Google Book Search for some books is wrong (for various reasons which I need not go into here), which I often found was the explanation for an isolated unexpected result.

Fifthly, when Google Book Search returns more than a page's worth of results, the number of results may go down significantly by the time you get to the last page.

Finally, and to me this is most perplexing, Google Book Search searches in March 2008 gave me over twice as many matches than in May 2008 using the same search criteria, so, for example, I got 438 matches for 'husband' and 956 matches for 'hufband' for the period 1790–1799 in March, but only 187 and 441 matches


The
NUMBER 5810.
PATRIOTIC FUND.
LLOYD's, SEPT. 6, 1803.
A T a GENERAL MEETING of the COMMITTEE for managing the PATRIOTIC FUND at Lloyd's, held this day in Merchant Seamen's Office,
Sir FRANCIS BARING, Bart. Chairman.
Amount fubfcribed, 3 per Cent. Conſols. 20,200l.
Amount of Subſcriptions - 145,638l. 14s 9d.
Adjourned to Tueſday, the 13th September, 1803, at 12 o'clock precisely.
FRANCIS BARING, Chairman.
Stephen Aiſley, Kenſington - 52. 10 0
The full and clear Receipts of one Night's Performance at the Theatre Royal, Haymarket, on the 20th of Aug. per George Colman, Eſq. 201 12 0
W. Yates, Bury, Lancaſter - 500 0 0
The Solicitor General, per Snow and Co. 105 0 0
J. M. - 1 1 0
J. S. P. Portſmouth 0 15 6
T. Prideaux - 21 0 0
Allen Tucker - 50 0 0
Fallett, Burnett and Sons - 105 0 0
Wm. Young Knight 21 0 0
Wm. Cooke, Exchequer, Weſtminſter 5. 0 0
Meſſrs. Manan Scott 10 10 0
The Levant Company 500. 0 0
Francis Turnly, Richmond, near Belfaſt 50 0 0
M. Raper, Stratford-place - 105 0 0
Mr. Walker, Harleybury, Hants, 25 guineas; Mrs. Walker 10 gs.; Miſs E. Walker 10 gs.; Miſs A. F. Walker 2 gs.; Miſs M. A. Walker 2 gs.; Miſs S. Walker 2 gs.
The
NUMBER 5811.
PATRIOTIC FUND.
LLOYD's, SEPT. 6, 1803.
A T a GENERAL MEETING of the COMMITTEE for managing the PATRIOTIC FUND at Lloyd's, held this Day in the Merchant Seamen's Office,
Sir FRANCIS BARING, Bart. Chairman.
Amount subscribed, 3 per Cent. Consols. 20,200l.
Amount of Subscriptions - 145,638l. 14s. 9d.
Adjourned to Tuesday, the 13th September, 1803, at 12 o'clock precisely.
FRANCIS BARING, Chairman.
His Grace the Abp. of Canterbury 500 0 0
The Worshipful Company of Clothworkers - 300 0 0
Henry Baldwin 25 0 0
Rev. Francis Fortescue, Knotsford - 10 10 0
Bruce Boswell, Iver Lodge - 50 0 0
Rev. Francis Drake, of Beverley 10 10 0
Rev. James Cox, Master of the Guildborough School 5 10 0
The Ushers and 27 Students of the same School - 5 10 0
Patrick Heafly - 30 0 0
The Mite of a Half-Pay Lieutenant 3 3 0
His Wife - 2 2 0
Three Sons under age 6 6 0
And 6 Daughters do. 3 3 0
Lord Wm. Gordon 100 0 0
Daniel Draper 52 10 0
Robert Stevenson 100 0 0
Capt. Rd. Williams, Royal Marines 10 10 0
J. and R. Colquhoun 52 10 0
Subscriptions received by Bennett and White, at the Bar of this House; and Letters for the Committee are desired to be addressed to Mr. Welsford, Secretary to the Patriotic Fund Committee, Lloyd's Coffee-house.

Figure 16: The Times Issues 5810 & 5811 (September 9th and 10th 1803). Compare the words *ſubſcribed*/*subscribed* and *Tueſday*/*Tuesday* in the first paragraph.

respectively for the same search when redone in May (nevertheless, the figures for March and May showed exactly the same trends for *husband* versus *huſband*). For consistency, the figures shown for 'husband/huſband' and 'ask/aſk' are those that I obtained in May 2008. (I may try redoing this experiment in a year's time — providing Google Book Search does not improve its OCR software to recognise *long s* in pre-19th century books — and see if the trends for *husband* versus *huſband* and *ask* versus *aſk* are roughly the same or not.)

14 And finally ...

If you have managed to get this far, you may well be interested in my brief, illustrated history of the *long s* (*The Long and the Short of the Letter S*), which to most people's surprise starts in Roman times.

And if the rules of *long s* are not enough for you, try out my *Rules for R Rotunda* (a post that I think needs some revision when I have the time).

◇ Andrew West
babelstone (at) gmail dot com
http://www.babelstone.co.uk/

Installing TEX Live 2010 on Ubuntu

Enrico Gregorio

Abstract

Is it possible to have TEX Live 2010 installed along with the Ubuntu version of the GNU/Linux operating system? Yes, and here's how it may be done.

1 Introduction

A major drawback of the TEX packaging on Ubuntu systems is that, due to a specific choice by the Debian developers, the maintenance program tlmgr is missing. On the one hand this protects the user from possible damages to the system; on the other hand, it prevents dynamic updates of the TEX distribution, in order to get bug fixes or to have available all the new functions that ship almost every day. Moreover, as of November 2010, the TEX Live provided by Ubuntu (by Debian, one should say) is still the 2009 edition, frozen as of October 2009.

We describe here a procedure to install TEX Live 2010 on a desktop Ubuntu system; this TEX Live will have its tlmgr for maintenance and updates. And, most important, it will not put anything dangerous in system directories. Only two (or three) files will be stored under `/etc`, in subdirectories already reserved for system customization.

To revert to using the system TEX, the entire TEX Live installation may be deleted from the system by a simple

```
$ rm -fr /usr/local/texlive/2010
```

and optionally by deleting the added files under `/etc`.

The procedure we'll describe can be adapted, with suitable changes, also to other distributions such as Fedora and OpenSUSE. Other Debian-based distributions should work the same as Ubuntu. We chose Ubuntu because it seems a frequently used distribution found on desktop GNU/Linux systems.

An Italian version of this paper appeared also on ArsTEXnica. My hope is that this article will become an installation program: it shouldn't be very difficult for a Unix guru to turn this procedure into a set of scripts for the various distributions.

2 A short introduction to the terminal

The procedure will require a certain familiarity with the terminal, that is, the interface to the command line. Anyone who hasn't the slightest idea of what the terminal is should stop reading here. However, it's not so difficult to copy the commands exactly as they are written here. There are many introductions to the command line, for example

`https://help.ubuntu.com/community/UsingTheTerminal`

In what follows, a line such as

```
$ ls -l
```

denotes a command that must be typed on the terminal, followed by a carriage return in order to execute it. The symbol `$` represents the system prompt, on your terminal it may be different, for example something like

```
enrico@ubuntu:~$
```

Normally, after these initial characters, there's a flashing box. Copy the commands starting after the space following the `$`. Other `$` characters that are not at the start of the line must be typed. Sometimes the commands will be too long to be printed on one line in this article; they will be rendered with

```
$ command a b c \
  d e f
```

The backslash denotes that the command continues on the following line. The spaces (or lack thereof) before the backslash are significant.

Answers by the system will be represented without the `$` symbol and underlined, for instance

```
bash: tix: command not found
```

says that the system would have executed the command tix, but it doesn't exist. The prefix `bash:` denotes the *shell* that's trying to execute the commands; ignore these details.

Almost always it's not necessary to copy completely the various pieces of a command line: press the TAB key and, if the completion is unique, the system will do it by itself.

Final advice: if your keyboard doesn't have a `~` key, find a way to type this character — and get, as soon as possible, a keyboard which has it.

► Notes for experienced users are introduced by this triangle. This paper assumes that the default shell is bash. Whoever uses a different shell qualifies as 'experienced user', and should be able to adapt the commands presented here to their particular shell(s).

3 Preliminaries

Install Perl-Tk and Perl-doc with Synaptic. Then open a terminal session and prepare a work directory, for example,

```
$ mkdir ~/texlive-install
$ cd ~/texlive-install
```

The latter command will put you inside the directory created with the former.

Now we'll briefly describe the two main ways to obtain TEX Live: the first works completely on-line, the second can also be done off-line.

4 Obtaining the distribution (on-line)

The simplest way to install TEX Live is through the Internet. Download the needed compressed archive by typing on the terminal

```
$ wget http://mirror.ctan.org/systems/\
  texlive/tlnet/install-tl-unx.tar.gz
```

Now you have to uncompress the downloaded file. The command is

```
$ tar zxf install-tl-unx.tar.gz
```

which will produce a new directory, where we'll go:

```
$ cd install-tl-20100914
```

The final part of the name is the date when the installation program has been produced, so it can be different: the completion feature mentioned before is what we need to find out the right suffix. Go to section 6.

5 Obtaining the distribution (off-line)

If your link to the Internet does not suffice, you can download an ISO image of the distribution, which is a file equivalent to a DVD. The Web address to contact is

```
http://mirror.ctan.org/systems/texlive/
Images/texlive2010.iso
```

It's a 1.9 GiB file; transfer it on a USB pen drive and copy it onto the machine where the installation is desired, or transform the ISO image to a DVD and insert it in the machine.

There is also a 'torrent': go to `http://www.tug.org/texlive/acquire-iso.html` and find the link to click; this should start the Transmission program that will download the image.

You can also get a physical DVD, either by joining a TEX user group or purchasing one separately: `http://www.tug.org/texlive/acquire-dvd.html`.

Ultimately, a double click on the ISO image allows access to the virtual (or physical) disk. Open a terminal session, create a work directory with

```
$ mkdir ~/texlive-install
```

and go to the `texlive` directory on the virtual (or physical) disk with something like

```
$ cd /cdrom/TeXLive/texlive
```

Instead of "`TeXLive`" it might be something else; use the automatic completion. Go to section 6.

6 Installing the distribution

Now you have to type the installation command:

```
$ sudo ./install-tl -gui \
  -repository http://mirror.ctan.org/\
  systems/texlive/tlnet
```

The system will ask for an administrator password and a window similar to that in figure 1 will pop up.

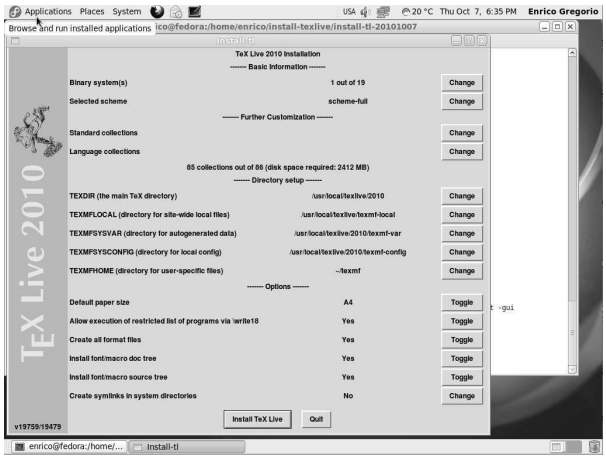


Figure 1: Installation window

If your default paper size is Letter, press the corresponding 'Toggle' button (the default for TEX Live is A4). At the bottom of the window, there's the Install TeX Live button. Press it and be confident that the installation will arrive at completion.

Do not modify the default setting for "Create symlinks in system directories": it *must* remain set to "No".

When the installation is finished, go to section 8.

7 If something goes wrong

If the installation doesn't finish, before retrying you must delete everything has been already written on your system:

```
$ cd /usr/local/texlive
$ ls
  2010        texmf-local
$ sudo rm -rf 2010
$ cd -
```

Then retry. Pay attention because the command rm is very dangerous, so carefully copy the commands. The output from the second command (ls) should be as shown, except possibly `texmf-local` may be missing. If the answer is different, check the first command and retype it correctly.

8 Touching up the installation

Now comes the difficult part: make the system know where to find the TEX distribution programs. Don't touch the standard distribution on Ubuntu, so you'll have no problem in installing programs that depend on it, such as Kile. There is some escape from this, but it requires installing the equivs package with Synaptic and doing some tricks that are very system dependent.

Go back to your work directory by

```
$ cd ~/texlive-install
```

and let's face up to the operating system. Give the mysterious commands

```
$ echo 'export PATH=/opt/texbin:${PATH}' \
  > texlive.sh
$ sudo cp texlive.sh /etc/profile.d/
$ sudo mkdir -p /opt
```

This creates a file `texlive.sh` containing the text we have written between single quotes and copies it into a system directory. The decisive step now requires a choice which depends on the hardware architecture of your machine; give one, and only one, of the following commands

```
$ sudo ln -s /usr/local/texlive/2010/bin/\
  i386-linux /opt/texbin
$ sudo ln -s /usr/local/texlive/2010/bin/\
  x86_64-linux /opt/texbin
$ sudo ln -s /usr/local/texlive/2010/bin/\
  powerpc-linux /opt/texbin
```

The author can't know which one: you must select the appropriate choice, depending on whether your machine is based on a 32-bit Intel (or AMD) processor, on a 64-bit x86 processor, or on a PowerPC chip. You can discover this by typing the command

```
$ ls /usr/local/texlive/2010/bin
```

which will answer with the required string.

▶ The experienced user might ask why we don't put inside `texlive.sh` the name of the directory with the executable programs, for example

```
export PATH=/usr/local/texlive/2010/\
  bin/i386-linux:${PATH}
```

The idea is that when the TeX Live 2011 ships out, it will be sufficient to type the command

```
$ sudo ln -s /usr/local/texlive/2011/bin/\
  i386-linux /opt/texbin
```

after installation, *without any other intervention*. Almost ... see later.

Now do a *logout*, because the system must digest the change. Then *login* again, open a terminal session and check that everything is ok; the command

```
$ which tex
```

should answer with

```
/opt/texbin/tex
```

If it's so, we are ready to update the distribution. Otherwise seek help from a guru.

There are two ways to use tlmgr; it's best to try both. The first starts tlmgr from the terminal. Type the command

```
$ gedit ~/.bashrc
```

and, in the window that opens, add, at the end,

```
# .bashrc addition for TeX Live
function sutlmgr () {
  if [[ -z "$@" ]]
  then
    sudo /opt/texbin/tlmgr -gui
  else
    sudo /opt/texbin/tlmgr "$@"
  fi
}
alias mktexlsr=
  'sudo /opt/texbin/mktexlsr'
alias updmap-sys=\
  'sudo /opt/texbin/updmap-sys'
alias fmtutil-sys=\
  'sudo /opt/texbin/fmtutil-sys'
```

The three dots represent what is already in the file `.bashrc`, which must not be modified.

▶ The experienced user might prefer to make the change in the file `.bash_aliases`.

Save the change with the proper menu entry, exit gedit and at the terminal type

```
$ . ~/.bashrc
$ sutlmgr
```

The tlmgr window will appear, after the system has asked for a password. From now on the sutlmgr command will start tlmgr with administrator privileges. Thus, a command such as

```
$ sutlmgr show --list xyz
```

directly executes the requested tlmgr action. With

```
$ texdoc tlmgr
```

you can access the documentation about tlmgr.

One may also create a small application on the desktop (this holds for the Gnome window manager; there may be something similar for KDE). Point the mouse on the desktop, press the right button and choose "Create Launcher ...". In the window that will show up, write '`TeX Live Manager`' in the 'Name' box, and in the 'Command' box type

```
gksu -d -S -D "TeX Live Manager"
  '/opt/texbin/tlmgr -gui'
```

(all on one line, please). After creating the application a double click on its icon will ask for an administrator password and launch tlmgr.

There's a last thing to do: make the system aware of the OpenType fonts provided along with TeX Live, in order to use them with XeTeX/(La)TeX by font name and not just file name.

```
$ sudo cp \
  $(kpsewhich -var-value TEXMFSYSVAR)\
  /fonts/conf/texlive-fontconfig.conf \
  /etc/fonts/conf.d/09-texlive.conf
$ sudo fc-cache -fsv
```

The commands for the procedure on a typical Ubuntu 10 installation are repeated in the final table 1; in italics are the parts that might be different; the underlined lines represent answers from the system, the one underlined and in italics represents what to type in place of `i386-linux` in the following line; the lines with actions written between angle brackets describe maneuvers that must be done outside the terminal.

9 OpenSUSE

The described procedure works with OpenSUSE, provided that Perl-Tk has been installed. There's no need to define a function to launch tlmgr, but you have to remember to use the options `-c` to su and `-E` to sudo. Thus call the maintenance programs as

```
$ su -c tlmgr -gui
$ sudo -E updmap-sys
$ sudo -E mktexlsr
```

10 Fedora

You can use the same procedure with Fedora, at least for version 13. The Perl-Tk module isn't included in the standard distribution. You can retrieve it from

```
http://koji.fedoraproject.org/koji/
buildinfo?buildID=151517
```

The file `/etc/profile.d/texlive.sh` must be different and contain the following text:

```
#!/bin/bash
if ! echo $PATH |\
  /bin/egrep -q "(^|:)/opt/texbin($|:)"
then
  PATH=/opt/texbin:$PATH
fi
```

Another difference is that in Fedora only `root` is the administrator, so the commands that in Ubuntu are prefixed with sudo must be typed without it, after having executed an su command.

11 Updating to TEX Live 2011

When the time comes, install the distribution as described in section 6. Then do

```
$ sudo ln -s /usr/local/texlive/2011/bin/\
  i386-linux /opt/texbin
$ sudo cp \
  $(kpsewhich -var-value TEXMFSYSVAR)\
  /fonts/conf/texlive-fontconfig.conf \
  /etc/fonts/conf.d/09-texlive.conf
$ sudo fc-cache -fsv
```

That's all (of course changing `i386-linux` if your architecture is different). No need to log out, no `texlive.sh` to write. Of course TEX Live 2011 has not yet been released, but we are ready for it.

Appendix

In this appendix we assume that TEX Live has been installed on Ubuntu, following the outlined procedure. Change the commands related to tlmgr if you run another distribution.

A Installing a personal package

Let's assume we need a LATEX package that's not present in TEX Live; this might happen due to licensing issues, or because we want to try an experimental version. There are two places where you can put the necessary files. First of all download the archive from where it is hosted (CTAN or somewhere else) and decompress it in a work directory. To give an example, the package will be padua and the directory will contain the files `README`, *padua*`.ins`, *padua*`.dtx`, and *padua*`.pdf` (the parts in italics represent the real name). Open a terminal session and type the following command:

```
$ tex padua.ins
```

If the file with extension `.ins` is not present, then

```
$ tex padua.dtx
```

will probably do the same job. In both cases the system will generate some files that we have to store in the correct place. Of course these are simple and generic instructions; some packages have a more complex structure and the author's directions should be followed, adapting them to what is found here.

Now you have to decide whether to use the *personal* tree or the *local* one. The main difference is that in the second case the package will be available to all users of the machine; of course you'll need administrator privileges to access the local tree.

The word 'tree' refers to a hierarchical directory structure, necessary to be accessed efficiently by the TEX system.

On GNU/Linux systems, the personal tree is rooted at `~/texmf`, that is, it's a set of subdirectories of your home. The local tree is rooted at `/usr/local/texlive/texmf-local`. Actually it's not necessary to know where they are. We can define a shorthand to stand for the location of the chosen tree; for the personal tree the trick is to run

```
$ Local=$(kpsewhich -var-value TEXMFHOME)
```

while for the local tree the trick is

```
$ Local=$(kpsewhich -var-value TEXMFLOCAL)
```

The TEX system is set up in such a way that it can know itself and the magic command is what stores in the variable `Local` the location of the chosen tree.

Let's now restart from where we stopped. We create the necessary directories and copy the files in them.

```
$ mkdir -p $Local/source/latex/padua
$ cp README padua.ins padua.dtx \
  $Local/source/latex/padua
$ mkdir -p $Local/doc/latex/padua
$ cp padua.pdf $Local/doc/latex/padua
$ mkdir -p $Local/tex/latex/padua
$ cp *.sty ... $Local/tex/latex/padua
```

That last line is intended to represent the files that should be seen by (LA)TEX itself, which are typically generated from the `dtx`.

B Installing a font family

There are many instruction sets around the world about how to install new fonts we have bought or found for free. Installing fonts in the personal tree is not recommended, because it requires a constant labour by the user when updates to TEX Live include something related to fonts.

It's best to follow the instructions contained in the booklet "The font installation guide" by Philipp Lehman, available in TEX Live with the terminal command

```
$ texdoc fontinstallationguide
```

These instructions, by the author's choice, end with the preparation of the needed files. Let's assume that the font family is called '*Padua*', with TEX family name *zpd*. As usual the part to be changed will be in italics. Lehman's procedure creates a bunch of files in the work directory, with various extensions:

```
.tfm .vf .pfb .afm .map .sty .fd
```

These files must be stored in the correct place in the TEX system hierarchy. The correct place is the already described *local tree*. Again we don't need to know where it is; let's define a shorthand, create the infrastructure, and store the files.

```
$ Local=$(kpsewhich -var-value TEXMFLOCAL)
$ sudo mkdir -p \
  $Local/fonts/{afm,tfm,type1,vf}/padua
$ sudo cp zpd*.afm $Local/fonts/afm/padua
$ sudo cp zpd*.tfm $Local/fonts/tfm/padua
$ sudo cp \
  zpd*.pfb $Local/fonts/type1/padua
$ sudo cp zpd*.vf $Local/fonts/vf/padua
$ sudo mkdir -p $Local/tex/latex/padua
$ sudo cp *.sty *.fd $Local/tex/latex/padua
$ sudo mkdir -p \
  $Local/fonts/map/dvips/padua
$ sudo cp padua.map \
  $Local/fonts/map/dvips/padua
$ mktexlsr
```

We have placed the furniture, now we must provide the TEX system with the door key. There are two cases: it's the first time we are installing a new font family, or we've already followed this procedure. In the first case we have to create a file and store it in a suitable place:

```
$ echo "Map padua.map" > updmap-local.cfg
$ mkdir -p $Local/web2c
$ sudo mv updmap-local.cfg $Local/web2c
$ sutlmgr generate --rebuild-sys updmap
```

In the second case, we only need to append a line to an already existing file:

```
$ cp $Local/web2c/updmap-local.cfg .
$ echo "Map padua.map" >> updmap-local.cfg
$ sudo mv updmap-local.cfg $Local/web2c
$ sutlmgr generate --rebuild-sys updmap
```

(The last action, as well as the call to mktexlsr, can be executed from the graphical interface of tlmgr.) By doing so we can be certain that the door key won't be lost with TEX Live updates. See also `http://www.tug.org/fonts/fontinstall.html` for a more verbose description of the steps.

If, by chance, also the OpenType versions of our new fonts are available, add the following couple of lines:

```
$ sudo mkdir -p $Local/fonts/opentype/padua
$ sudo cp *.otf $Local/fonts/opentype/padua
```

to the similar ones seen before. Similarly, by changing `opentype` with `truetype` if the font files have extension `.ttf`.

If it's the first time you add OpenType, TrueType or Type 1 fonts to the local tree, you have to make the new location known to the system. Do

```
$ cp /etc/fonts/conf.d/09-texlive.conf \
  09-texlive-local.conf
$ gedit 09-texlive-local.conf
```

In the window that will appear, change all the strings '`2010/texmf-dist`' into '`texmf-local`'. Save and type in the terminal

```
$ sudo mv 09-texlive-local.conf \
  /etc/font/conf.d
$ sudo fc-cache -fsv
```

Now XE(LA)TEX will also be able to access the new fonts. Only the fc-cache call is needed if you already created the 'local' configuration file.

◇ Enrico Gregorio
Dipartimento di Informatica
Università di Verona
enrico dot gregorio (at) univr
dot it

Table 1: The whole procedure for Ubuntu

⟨*Install* Perl-Tk *and* Perl-doc *with* Synaptic⟩
⟨*Start terminal session*⟩
```
$ mkdir ~/texlive-install
$ cd ~/texlive-install
$ wget http://mirror.ctan.org/systems/texlive/tlnet/install-tl-unx.tar.gz
$ tar zxf install-tl-unx.tar.gz
$ cd install-tl-20100914
$ sudo ./install-tl -gui -repository http://mirror.ctan.org/systems/texlive/tlnet
```
⟨*Press ''Toggle'' for the paper format if you need to*⟩
⟨*Press ''Install TeX Live''*⟩
⟨*Wait until the installation is complete; drink a coffee, maybe two*⟩
⟨*Press ''End''*⟩
```
$ cd ~/texlive-install
$ echo 'export PATH=/opt/texbin:${PATH}' > texlive.sh
$ sudo cp texlive.sh /etc/profile.d/
$ sudo mkdir -p /opt
$ ls /usr/local/texlive/2010/bin
  i386-linux
$ sudo ln -s /usr/local/texlive/2010/bin/i386-linux /opt/texbin
```
⟨*Do a logout*⟩
⟨*After login, open a terminal*⟩
```
$ which tex
  /opt/texbin/tex
```
⟨*If the answer is different, cry out 'Help'*⟩
```
$ gedit ~/.bashrc
```
⟨*Append to the file*⟩
```
    # Additions for TeX Live
    function sutlmgr () {
      if [[ -z "$@" ]]
      then
        sudo /opt/texbin/tlmgr -gui
      else
        sudo /opt/texbin/tlmgr "$@"
      fi
    }
    alias mktexlsr='sudo /opt/texbin/mktexlsr'
    alias updmap-sys='sudo /opt/texbin/updmap-sys'
    alias fmtutil-sys='sudo /opt/texbin/fmtutil-sys'
```
⟨*Save and exit from* gedit⟩
```
$ sudo cp $(kpsewhich -var-value TEXMFSYSVAR)/fonts/conf/texlive-fontconfig.conf \
  /etc/fonts/conf.d/09-texlive.conf
$ sudo fc-cache -fsv
```
⟨*Relax and enjoy TEX Live 2010*⟩

Notes.
(1) The date `20100914` is an example, it will likely be different.
(2) `i386-linux` corresponds to one of the possible architectures (a.k.a. platforms);
it may also be `x86_64-linux` or, less probably, `powerpc-linux`.

tlcontrib.metatex.org: A complement to TEX Live

Taco Hoekwater

Abstract

TLContrib is a distribution and associated web site that hosts contributed, supplementary packages for TEX Live.

The packages on TLContrib are those not distributed inside TEX Live proper for one or several of the following reasons: because they are not free software according to the FSF guidelines, because they contain an executable update, because they are not available on CTAN, or because they represent an intermediate release for testing.

Anything related to TEX that cannot be in TEX Live but can still legally be distributed over the Internet can have its place on TLContrib.

Keywords: TEX Live, TLContrib, distribution, contribution, packages

1 Introduction

Many readers will be familiar with TEX Live as an easy way to install TEX. This distribution provides a comprehensive TEX system with binaries for most flavors of Unix, including GNU/Linux and Mac OS X, and also Windows. It includes all the major TEX-related programs, macro packages, and fonts that are free software, including support for many languages around the world. The current version is TEX Live 2010.

TEX Live is distributed on DVD by most of the local TEX user groups, but it also allows for continuous package updates over the Internet using the `tlmgr` program.

TEX Live is a wonderful tool, but there are a few considerations to be aware of:

- it contains only FSF-defined 'free software' packages
- it uses CTAN as its primary source for packages
- it does not make interim executable updates
- it is not a suitable medium for package test releases

Each of these limitations has a perfectly reasonable cause:

- The TEX Live maintainers agree (at least for the purposes of working on TEX Live) with the principles and philosophy of the free software movement. Therefore they follow the FSF guidelines on licensing.
- It is good for the TEX community if CTAN is as complete as possible. That gives users one place to look, for instance. Also, it makes it more likely for separate distributions like TEX Live and MiKTEX to be consistent with each other. By using CTAN as the primary package source, TEX Live promotes the use of CTAN.

  A secondary reason for the use of CTAN is that creating a large distribution like TEX Live takes a lot of work, and the number of volunteers is limited. Having a single place to check for new package updates is a lot easier, because this process can be automated to a large extent. Using many separate sources would make this task much more complicated.
- TEX Live ships binaries for 19 different computer platforms, and something like 300 binaries need to be compiled for each of those. Coordinating the task of preparing these binaries is a major effort.
- Because TEX Live is not just a network installation, but also shipped on DVD, it is important that the included packages and binaries are as stable as possible. After all, there is no guarantee that the DVD users will *ever* update their system after the initial installation.

Nevertheless, the limitations of TEX Live mean that there is room for extension. This is the reason for the existence of TLContrib.[1]

On TLContrib, anything that is freely distributable is acceptable, so packages that are not on CTAN are fine, and TLContrib can and will contain updates to executables (though not necessarily for all platforms).

This is possible because the two major limitations of TEX Live do not exist in TLContrib. Firstly, TLContrib is a network-only distribution without the limitations introduced by the physical medium. Secondly, the problem of lack of human resources is solved by offloading the burden of creating and maintaining packages to the actual package maintainers.

Before going on to explain how to use TLContrib, it is important to note the following:

- TLContrib is *not* a full TEX Live repository: it is a complement and contains only its own packages. This means TLContrib can only be used as a secondary repository on top of an existing TEX Live installation.
- TLContrib is not maintained by the TEX Live team: the responsibility for the actual packages lies with the package maintainers themselves,

Editor's note: This article appeared originally (in slightly different form) in *MAPS* 41, fall 2010.

[1] The web site for TLContrib is `http://tlcontrib.metatex.org/`

and the server maintenance is handled by yours truly.

There is no competition between TLContrib and TEX Live, but as one of the goals of TLContrib is to ease the workload of the TEX Live team, it would not make much sense for them to be the actual maintainers. For this reason there is a separate mailing list dedicated to TLContrib.[2] Please address your questions related to packages obtained from TLContrib there, and not on the regular TEX Live list.

2 Using TLContrib as a distribution

First things first: before attempting to use TLContrib, make sure that you have the latest (network) update of TEX Live 2010, and in particular that you run the latest `tlmgr`. During the development of TLContrib, a small number of incompatibilities were found in the `tlmgr` as distributed on the DVD that have since been fixed. Furthermore, the current version of TLContrib works only with TEX Live 2010 and not for any earlier versions of TEX Live.

And a warning: Executable packages are not necessarily available for all platforms on TLContrib. Unfortunately, it appears that the current TEX Live update manager is not smart enough to correctly detect versioning in dependencies. In practice, this means that you should not update packages that depend on executable package updates *unless* the actual executable package update is also available on TLContrib for your platform.

Keeping the above in mind, in order to use TLContrib as an extra repository in the TEX Live 2010 package manager (`tlmgr`), there are two options, depending on whether you prefer to use the command line version or the GUI version of the TEX Live 2010 package manager.

2.1 Graphical interface usage

In the GUI version of the package manager, select the menu item Load other repository ... from within the tlmgr menu. Set the value to

```
http://tlcontrib.metatex.org/2010
```

There is currently no way to save this setting.

Besides not being able to save the TLContrib setting, when using the graphical user interface it is not always easy to see whether executable package updates are available. For this reason you should consider using the command line version of the package manager for use with TLContrib, even if you are accustomed to using the GUI interface.

[2] The mailman page for the mailing list is `http://www.ntg.nl/cgi-bin/mailman/listinfo/tlcontrib`.

2.2 Command line usage

The simplest approach is to just start `tlmgr` from the command line with an extra option:

```
$ tlmgr --repository \
  http://tlcontrib.metatex.org/2010
```

If you plan to use TLContrib regularly, it makes sense to define a shell alias to save you some typing (this trick is courtesy of Will Robertson). To do so, put the following into your `.bash_profile` or equivalent (this has to be on a single line):

```
alias tlc="tlmgr --repository
           http://tlcontrib.metatex.org/2010"
```

You can then view what is available in the TLContrib repository with standard `tlmgr` commands such as

```
$ tlc list
```

to see what is currently available for installation. Packages can be updated to their pre-release versions by typing, say,

```
$ tlc update siunitx
```

If an update performed in this way 'goes bad' and you'd like to revert to the official release, execute

```
$ tlmgr install siunitx --reinstall
```

and things will be back to normal.

3 Using TLContrib for distribution

The rest of this article describes further details important for a package maintainer aiming to use TLContrib for distribution.

Before you decide to add a package to TLContrib, please bear this in mind:

- It is not the intention of TLContrib to replace either TEX Live or CTAN: if a package is not blocked from TEX Live for one of the reasons mentioned earlier, and can be made available on TEX Live or CTAN, then it should not be part of TLContrib at all.

In order to be able to upload packages to TLContrib, you have to be a registered user. You can register as a user via the TLContrib web site, and, not by coincidence, this is also the place where you create new packages and package releases.

After registration is complete, you can log into TLContrib by following the member section link.

If you do upload a package to TLContrib, please also subscribe to the TLContrib mailing list, because any questions about your package are likely to be made there.

3.1 Package creation example

This quick start guide uses an update of the `context-lettrine`

package as an example of how to create a package. In the following, you need to replace `context-lettrine` by the actual package name that *you* are updating, of course.

Besides being logged in to TLContrib, the first thing you need to do is to create your updated package source. In this case, the easiest way is to start from the current TEX Live version, so first you have to fetch the current `context-lettrine` archive(s) from the network distribution of TEX Live. The base URL is: `http://www.ctan.org/tex-archive/systems/texlive/tlnet/archive`.

In fact, for this example, there are two archives to be downloaded:

```
context-lettrine.tar.xz
context-lettrine.doc.tar.xz
```

For some TEX Live packages there is even a third archive file named ⟨*package*⟩`.source.tar.xz`. This is because the distribution system of both TEX Live and TLContrib splits the contribution into run-time files, documentation files, and source files. Users can ask the installer not to install the last two file types to save on disk space and network traffic.

You have to create a single local archive file with the combined and updated content of the two downloaded archives. After extracting both `tar.xz` files in the same directory, you will have a tree structure that looks like this:

```
doc/
  context/
    third/
      lettrine/
        lettrine-doc.pdf
        lettrine-doc.tex
        W.pdf
tex/
  context/
    interface/
      third/
        lettrine.xml
    third/
      lettrine/
        t-lettrine.tex
tlpkg/
  tlpobj/
    context-lettrine.doc.tlpobj
    context-lettrine.tlpobj
```

First, delete the whole `tlpkg` sub-tree. The `tlpobj` files contain meta-data specific to each particular revision of a package, and the information in the downloaded version of these files will henceforth be no longer applicable. New versions of the `tlpobj` files will be generated automatically by TLContrib's distribution creation tool.

You may now update the other files in the tree, and create the archive file (the acceptable formats are `tar.gz`, `tar.xz`, and `zip`). Please read the next section, 'About package sources', carefully before finalizing the archive.

The TLContrib `context-lettrine` package will use the newly created archive as source for the package, so make doubly sure you use the right files. Starting with the existing TEX Live package archive(s) is just so you get an idea of what goes where: sometimes TEX Live packages contain more files and symbolic links than you initially expect. You can build the source package completely from scratch if you want, but it is easy to forget files if you don't check.

Incidentally, while the base name of the local archive file does not matter, you have to make sure that the extension is `.tar.gz`, `.tar.xz`, or `zip`, otherwise the upload will fail.

Now go to `http://tlcontrib.metatex.org` in your browser, log in, and click new package. As the new package is an update to TEX Live, make sure you select that option, and the proper package name from the drop-down (`context-lettrine`).

In the next screen, most of the needed input will be automatically filled in for you, based on the current TEX Live revision of the package.

Edit the rest of the input form to have a proper version and set the source to File upload. Its value has to be set to the new archive that was created earlier. Adjust the Release state drop-down so it is set to public. It is also wise to check the license field, for it does not always import correctly due to database mismatches.

Then press submit new revision, verify the upload, and submit again to finalize the new package.

Assuming all went well, all that is needed for now is to wait until the hour has passed: your package should be available from the TLContrib repository after that.

The TLContrib distribution system works asynchronously: the front-end data that you as a package maintainer can create and modify is exported to the user-side TLContrib repository by a cron job that runs independent of the actual web site. Currently this cron job runs hourly, on the hour.

3.2 About package sources

Please note: Currently only the `tar.gz`, `tar.xz`, and `zip` archive formats are supported in the File upload and HTTP URL methods, and there are further strict requirements on the archive itself:

For a non-executable package, it should contain a complete TDS (TEX Directory Structure; see `http://www.tug.org/tds/`) sub-tree. In TEX Live,

normally all macro files go under `texmf-dist`, and, in that case, this directory level can be skipped in the archive (it will be added automatically by the TLContrib publication system). Be advised that, in general, uploading a CTAN zipped folder will *not* work, because CTAN packages are almost never in TDS format.

For an executable package, you can also use the TDS layout (with the binaries in `bin/$ARCH/`), but if you only have files inside the binaries folder, you can skip the directory structure completely: in this case, the TLContrib publication system will automatically add the needed structure.

Make sure that your archive contains only files that belong to your package, and especially that it does not accidentally overwrite files owned by other packages.

Also, check twice that the archive contains only files that belong in the TDS: Delete backup files, and remove any special files that may have been added by the operating system (Mac OS X especially has a very bad habit of adding sub-directories for its Finder that really do not belong in the package).

It is not always simple to guess what should go into a TEX Live update package. If you are building such an updated package, it is always wise to start from the existing TEX Live sources.

TLContrib accepts no responsibility for package contents: the system does run some sanity checks, but ultimately, you as maintainer are responsible for creating a correctly functioning package. Badly behaving or non-working packages will be removed on executive decision by the TLContrib maintainer(s) without prior notice.

3.3 Package creation in detail

When you create a new package, a short wizard will help present itself to help you set up the package type. There are two types of packages: those that are updates of existing TEX Live packages, and those that are standalone. The wizard screen presents you the choice between these two types, and a drop-down listing TEX Live packages. The list of existing TEX Live packages is updated daily. Once this decision is made, it becomes fixed forever: the Id field of a package cannot be edited afterwards.

The Id field is the internal identifier of the package. An Id should consist of a single 'word', with a length of at least two characters, that contains only alphanumerics, dashes, and underscores. It can optionally followed by a platform identifier, which is then separated from the first part by a single dot.

Also note that when Release state becomes public (as explained below), it will no longer be possible to edit that particular release of the package. All further edits will force the creation of a new release, with a new revision id, and needing new sources.

Yet another note: If you intend to create an executable package, you have to be really sure you know what you are doing. Creating portable binaries for any platform is far from trivial. Paraphrasing Norbert Preining from the TLContrib mailing list:

> If you have NO experience with compiling, preparing binaries for various platforms, distributing, etc., JUST DO NOT GO THERE!

Macro packages are much easier; for those you need only a good understanding of how the TDS works.

3.4 Package editing

After the initial New package wizard screen, or after pressing Edit in the your package list for pre-existing packages, you will be presented with a fairly large edit screen.

During the initial TLContrib package creation process, if the package is updating an existing TEX Live package, certain fields will have been filled in automatically from the TEX Live package database. Otherwise you will have to fill in everything yourself.

Title
: The human-readable name of your package.

Description
: This is a description in a few sentences of what the package does.

Package type
: Even though the drop-down is long, really there are only two choices in the drop-down: A package is either a Macro package, or na Executable package. The distinction is important because the required package source structure is different for each of the two types, as explained below.

TLMGR directives
: A list of `tlmgr` directives, e.g. addMap or addFormat. A better interface is planned, but, for the moment, you have to make sure you know what you are doing. Have a look at the existing TEX Live package database (the file named `texlive.tlpdb`) for examples.

  You only have to specify the directives; do not add execute at the start.

TL dependencies
: Package Ids of other TEX Live packages on which this package depends, one per line. Unless you know exactly what is needed, it is probably best to leave this field blank, but in any case:

  You only have to specify the package Ids; do not add depend at the start. If your package

depends on an executable package, for example luatex, write the Id as luatex.ARCH. Doing so will make `tlmgr` automatically select the appropriate executable package for the user's platform.

TL postactions
: A list of `tlmgr` post-install actions, e.g. shortcut or fileassoc. A better interface for this is also planned, but, for the moment, you have to make sure you know what you are doing here as well. Have a look at the existing TeX Live package database (`texlive.tlpdb`) for examples.

  You only have to specify the actions, do not add postaction at the start.

License
: Pick one from the two drop-downs, and set the radio button accordingly. If you need to use Other free license or Other non-free license, please drop me an email. I am sure the list is incomplete. In this context, Free means: according to the Debian Free Software Guidelines.

Log message
: This field is just for release notes: it will not be exported to the TLContrib repository. The SVN URL and GIT URL methods will automatically refill in this field with the remote revision and log message. For other source methods, you can fill in whatever seems appropriate.

Release state
: Only packages that are public are exported, but this also has side-effects. Once the Release state is public, it is no longer possible to edit a package release on the spot. Submitting the form in that case will always create a new release.

  On edits, you will see some extra information: Synch state and rev. The first is the current status of a package release with respect to the published repository, the second is the revision number that has been assigned to this release.

Version
: This is the user-visible version field.

Source
: Here things get interesting. There are five ways to put the source of a package release into the database, as explained in the next sections.

  - As previous revision
    If you are editing an already existing package, then it is possible to re-use the uploaded source from the revision you are editing as the source for the new revision that will be created.
  - File upload
    Upload of a local archive file via CGI. Be warned that if there are other errors in your form, you will have to re-select the local file after fixing those other errors. Contrary to the other fields, local file selection is not persistent across form submits.
  - HTTP URL
    This asks the system to do a `wget` of an archive file on a specific URL, which could be either HTTP or FTP. If you need remote log-in information to access the file, please encode the user name and password in the URL, exactly as you would do when using `wget` on the command line.
  - SVN URL
    This asks the system to do an `svn checkout` of a specific URL. In this case, you may also need SVN Username and SVN Password. Also, some repositories may need `anonymous` as the user name for anonymous access. The top-level checkout folder will be stripped away before creating the package. This is so that you can give, e.g. `http://foundry.supelec.fr/svn/metapost/trunk/texmf/`, as the URL without getting an extra directory level.
  - GIT URL
    This asks the system to do a `git clone` of a specific URL. It is very similar to SVN URL, just using a different versioning system. In this case, you may also need GIT Branch.

Please verify package contents
: The first time the edit form is loaded, this will only display a message, but after the initial submit (assuming everything else went well), it will display the full list of files that will become the source of your package.

  Please check this list carefully! TLContrib does run some tests on the package contents and will refuse to accept package sources that are horribly wrong, but it does not check the actual contents of any of the files, and of course it cannot test for every possible problem.

3.5 Package transfer

It is possible for the maintainer of a package to transfer the package to another user completely. To do so, follow the Share link in your package list. See the help text in that form for details.

3.6 Package sharing

It is also possible for the maintainer of a package to share the package maintenance with other users.

To set up package sharing for a package you maintain, follow the Share link in your package list. See the help text in that form for details.

When someone else has shared a package with you, then you will see new entries in your package list. These will have the user id of the actual package maintainer added after the Date field. You can edit such packages (and thus create new revisions), but the new revisions will become property of the actual package maintainer.

In other words: a package can have only one actual maintainer, and that maintainer is responsible for all revisions of the package. However, the maintainer can allow other users to help with the actual creation of new revisions.

3.7 Package deletion

In the list of your packages and in the view screen of one of your package releases, there are two links that delete items:

- Del / Delete revision
  This link deletes a single revision of a package.
- Delete package (in view screen only)
  This link removes a whole package completely, including all revisions of it.

Both links show a confirmation screen first.

3.8 Remote revision creation (advanced usage)

Once a package has been created (there must be at least one revision record present already), and under the conditions that it has a source method of HTTP URL, SVN URL, or GIT URL, it is possible to submit a new revision by fetching a special URL from a remote location or script. Using this method, there is no need to be logged in at all.

The URL template looks like this:

```
http://tlcontrib.metatex.org
  /cgi-bin/package.cgi/action=notify
  /key=⟨key⟩
  /check=⟨md5⟩
  ?version=⟨version⟩
```

Please note that `version` is preceded by a question mark, but everything else is separated by slashes, and, of course, the actual URL should be a single line, without any spaces. All three fields are required. The three special fields have to be filled in like this:

⟨*key*⟩
: This is the Id of the package. For example, we'll use `luatex.i386-linux`.

⟨*md5*⟩
: This is a constructed checksum, created as follows: it is the hexadecimal representation of the md5 checksum of the string created by combining your userid, your password, and the new version string, separated by slashes.

  For example, let's assume that your userid is `taco` and your password is `test`, and that the new release that you are trying to create has version `0.64.0`. On a Unix command line, the checksum can then be calculated like this:

```
$ echo taco/test/0.64.0  md5sum
c704f499e086e0d54fca36fb0abc973e  -
```

  The value of ⟨*md5*⟩ is therefore:

```
c704f499e086e0d54fca36fb0abc973e
```

⟨*version*⟩
: This is the version field of the new release.

  Note: if this contains spaces or other characters that cannot be used in URLs, then you either have to escape the version string in the URL, or use POST instead of GET. In any case, do not escape the version while calculating the checksum string.

  There is no need to do any URL escaping in our case, so our value of ⟨*version*⟩ will be `0.64.0`.

Using the example variables given above, the final URL that would have to be accessed is (again without line breaks or spaces):

```
http://tlcontrib.metatex.org
  /cgi-bin/package.cgi/action=notify/
  key=luatex.i386-linux
  /check=c704f499e086e0d54fca36fb0abc973e
  ?version=0.64.0
```

Accessing this URL will cause TLContrib to fetch the HTTP or SVN or GIT URL source in the package's top-level revision (regardless of what its publication state is), and create a new revision based on the fetched file(s) and the supplied version string. All other fields will remain exactly the same as in the original top-level revision.

This new package revision will appear in the web interface just like any other revision, there is nothing special about it other than what is already mentioned.

4 Final remark

TLContrib is a fairly new project, and some improvements are definitely possible, especially in the edit forms on the web site. But I hope that even in the current state, it will be a useful addition to the whole TEX Live experience.

◇ Taco Hoekwater
http://tlcontrib.metatex.org

LuaTEX: What it takes to make a paragraph

Paul Isambert

Introduction

The road that leads from an input to an output document is rather eventful: bytes must be read, interpreted, executed, glyphs must be created, lines must be measured ... With LuaTEX those events can be monitored and their courses can be bent; this happens in callbacks, points in TEX's processing where custom code can be inserted. This paper will look at the callbacks involved from reading an input line to releasing the product of the paragraph builder to a vertical list. The callbacks that we will be looking at are the following:

`process_input_buffer`
: How TEX reads each input line.

`token_filter`
: How TEX deals with tokens.

`hyphenate`
: Where discretionaries are inserted.

`ligaturing`
: Where ligatures happen.

`kerning`
: Where font kerns are inserted.

`pre_linebreak_filter`
: Before the paragraph is built.

`linebreak_filter`
: Where the paragraph is built.

`post_linebreak_filter`
: After the paragraph is built.

Actually, a few more callbacks are involved, but these are most relevant to paragraph building.

Reading input lines

The `process_input_buffer` callback is executed when TEX needs an input line; the argument passed to the callback is the input line itself, and another line, possibly the same, should be returned. By default, nothing happens, and what TEX reads is what you type in your document.

A line in this context is actually a Lua string; hence what the callback is supposed to do is string manipulation. Besides, one should remember that this line hasn't been processed at all; for instance, material after a comment sign hasn't been removed, and multiple spaces haven't been reduced to one space; of course, escape characters followed by letters haven't been lumped into control sequences. In other words, the string is exactly the input line.

Can anything useful be done by manipulating input lines? Yes, in fact the `process_input_buffer` callback proves invaluable. Here I'll address two major uses: encoding and verbatim text.

Using any encoding. Unlike its elder brothers, LuaTEX is quite intolerant when it comes to encodings: it accepts UTF-8 and nothing else. Any sequence of bytes that does not denote a valid UTF-8 character makes it complain. Fortunately, ASCII is a subset of UTF-8, thus LuaTEX understands most older documents. For other encodings, however, input lines must be converted to UTF-8 before LuaTEX reads them. One main use of the `process_input_buffer` callback is thus to perform the conversion.

Converting a string involves the following steps (I'll restrict myself to 8-bit encodings here): mapping a byte to the character it denotes, more precisely to its numerical representation in Unicode; then turning that representation into the appropriate sequence of bytes. If the source encoding is Latin-1, the first part of this process is straightforward, because characters in Latin-1 have the same numerical representations as in Unicode. As for the second part, it is automatically done by the `slnunicode` Lua library (included in LuaTEX). Hence, here's some simple code that allows processing of documents encoded in Latin-1.

```
local function convert_char (ch)
  return unicode.utf8.char(string.byte(ch))
end
local function convert (line)
  return string.gsub(line, ".", convert_char)
end
callback.register("process_input_buffer", convert)
```

Each input line is passed to `convert`, which returns a version of that line where each byte has been replaced by one or more bytes to denote the same character in UTF-8. The Lua functions work as follows: `string.gsub` returns its first argument with each occurrence of its second argument replaced with the return value of its third argument (to which each match is passed). Since a dot represents all characters (i.e. all bytes, as far as

The code in this paper has been written and tested with the latest build of LuaTEX. The reader probably uses the version released with the latest TEX Live or MikTEX distributions, and differences might occur. More recent versions can be regularly downloaded from TLContrib, a companion repository which hosts material that doesn't make it to TEX Live for whatever reason. Bleeding-edge LuaTEX can also be built from the sources.

Lua is concerned), the entire string is processed piecewise; each character is turned into a numerical value thanks to `string.byte`, and this numerical value is turned back to one or more bytes denoting the same character in UTF-8.

What if the encoding one wants to use isn't Latin-1 but, say, Latin-3 (used to typeset Turkish, Maltese and Esperanto)? Then one has to map the number returned by `string.byte` to the right Unicode value. This is best done with a table in Lua: each cell is indexed by a number m between 0 and 255 and contains a number n such that character c is represented by m in Latin-3 and n in Unicode. For instance (numbers are given in hexadecimal form by prefixing them with `0x`):

```
latin3_table = { [0] = 0x0000, 0x0001, 0x0002,
  ...
  0x00FB, 0x00FC, 0x016D, 0x015D, 0x02D9}
```

This is the beginning and end of a table mapping Latin-3 to Unicode. At the beginning, m and n are equal, because all Latin-x encodings include ASCII. In the end, however, m and n differ. For instance, 'ŭ' is 253 in Latin-3 and 0x016D (365) in Unicode. Note that only index 0 needs to be explicitly specified (because Lua tables starts at 1 by default), all following entries are assigned to the right indexes.

Now it suffices to modify the `convert_char` function as follows to write in Latin-3:

```
local function convert_char (ch)
  return unicode.utf8.char
    (latin3_table[string.byte(ch)])
end
```

Verbatim text. One of the most arcane areas of TEX is catcode management. This becomes most important when one wants to print verbatim text, i.e. code that TEX should read as characters to be typeset only, with no special characters, and things turn definitely dirty when one wants to typeset a piece of code and execute it too (one generally has to use an external file). With the `process_input_buffer` callback, those limitations vanish: the lines we would normally pass to TEX can be stored and used in various ways afterward. Here's some basic code to do the trick; it involves another LuaTEX feature, catcode tables.

The general plan is as follows: some starting command, say `\Verbatim`, registers a function in the `process_input_buffer`, which stores lines in a table until it is told to unregister itself by way of a special line, e.g. a line containing only `\Endverbatim`. Then the table can be accessed and the lines printed or executed. The Lua side follows. (About the `\noexpand` in `store_lines`: we're assuming this Lua code is read via `\directlua` and not in a separate Lua file; if the latter is the case, then remove the `\noexpand`. It is used here to avoid having `\directlua` expand `\\`.)

```
local verb_table
local function store_lines (str)
  if str == "\noexpand\\Endverbatim" then
    callback.register("process_input_buffer",nil)
  else
    table.insert(verb_table, str)
  end
  return ""
end
function register_verbatim ()
  verb_table = {}
  callback.register("process_input_buffer",
      store_lines)
end
function print_lines (catcode)
  if catcode then
    tex.print(catcode, verb_table)
  else
    tex.print(verb_table)
  end
end
```

The `store_lines` function adds each line to a table, unless the line contains only `\Endverbatim` (a regular expression could also be used to allow more sophisticated end-of-verbatims), in which case it removes itself from the callback; most importantly, it returns an empty string, because if it returned nothing then LuaTEX would proceed as if the callback had never happened and pass the original line. The `register_verbatim` function only resets the table and registers the previous function; it is not `local` because we'll use it in a TEX macro presently. Finally, the `print_lines` uses `tex.print` to make TEX read the lines; a catcode table number can be used, in which case those lines (and only those lines) will be read with the associated catcode regime. Before discussing catcode tables, here are the relevant TEX macros:

```
\def\Verbatim{%
  \directlua{register_verbatim()}%
}
\def\useverbatim{%
  \directlua{print_lines()}%
}
\def\printverbatim{%
  \bgroup\parindent=0pt \tt
  \directlua{print_lines(1)}
  \egroup
}
```

They are reasonably straightforward: `\Verbatim` launches the main Lua function, `\useverbatim`

reads the lines, while \printverbatim also reads them but with catcode table 1 and a typewriter font, as is customary to print code. The latter macro could also be launched automatically when store_lines is finished.

What is a catcode table, then? As its name indicates, it is a table that stores catcodes, more precisely the catcodes in use when it was created. It can then be called to switch to those catcodes. To create and use catcode table 1 in the code above, the following (or similar) should be performed:

```
\def\createcatcodes{\bgroup
  \catcode`\\=12 \catcode`\{=12 \catcode`\}=12
  \catcode`\$=12 \catcode`\&=12 \catcode`\^^M=13
  \catcode`\#=12 \catcode`\^=12 \catcode`\_=12
  \catcode`\ =13 \catcode`\~=12 \catcode`\%=12
  \savecatcodetable 1
\egroup}
\createcatcodes
```

The \savecatcodetable primitive saves the current catcodes in the table denoted by the number; in this case it stores the customary verbatim catcodes. Note that a common difficulty of traditional verbatim is avoided here: suppose the user has defined some character as active; then when printing code s/he must make sure that the character is assigned a default (printable) catcode, otherwise it might be executed when it should be typeset. Here this can't happen: the character (supposedly) has a normal catcode, so when table 1 is called it will be treated with that catcode, and not as an active character.

Once defined, a catcode table can be switched with \catcodetable followed by a number, or they can be used in Lua with tex.print and similar functions, as above.

As usual, we have set space and end-of-line to active characters in our table 1; we should then define them accordingly, although there's nothing new here:

```
\def\Space{ }
\bgroup
\catcode`\^^M=13\gdef^^M{\quitvmode\par}%
\catcode`\ = 13\gdef {\quitvmode\Space}%
\egroup
```

Now, after

```
\Verbatim
\def\luatex{%
  Lua\kern-.01em\TeX
  }%
\Endverbatim
```

one can use \printverbatim to typeset the code and \useverbatim to define \luatex to LuaTEX. The approach can be refined: for instance, here each new verbatim text erases the preceding one, but one could assign the stored material to tables accessible with a name, and \printverbatim and \useverbatim could take an argument to refer to a specific piece of code; other catcode tables could also be used, with both macros (and not only \printverbatim). Also, when typesetting, the lines could be interspersed with macros obeying the normal catcode regime (thanks to successive calls to tex.print, or rather tex.sprint, which processes its material as if it were in the middle of a line), and the text could be acted on.

Reading tokens

Now our line has been processed, and TEX must read its contents. What is read might actually be quite different from what the previous callback has returned, because some familiar operations have also taken place: material after a comment sign has been discarded, end-of-line characters have been turned to space, blank lines to \par, escape characters and letters have been lumped into control sequences, multiple spaces have been reduced to one ... What TEX reads are tokens, and what tokens are read is decided by the token_filter callback.

Nothing is passed to the callback: it must fetch the next token and pass it (or not) to TEX. To do so, the token.get_next function is available, which, as its name indicates, gets the next token from the input (either the source document or resulting from macro expansion).

In LuaTEX, a token is represented as a table with three entries containing numbers: entry 1 is the command code, which roughly tells TEX what to do. For instance, letters have command code 11 (not coincidentally equivalent to their catcode), whereas a { has command code 1: TEX is supposed to behave differently in each case. Most other command codes (there are 138 of them for the moment) denote primitives (the curious reader can take a look at the first lines of the luatoken.w file in the LuaTEX source). Entry 2 is the command modifier: it distinguishes tokens with the same entry 1: for letters and 'others', the command modifier is the character code; if the token is a command, it specifies its behavior: for instance, all conditionals have the same entry 1 but differ in entry 2. Finally, entry 3 points into the equivalence table for commands, and is 0 otherwise.

To illustrate the token_filter callback, let's address an old issue in TEX: verbatim text as argument to a command. It is, traditionally, impossible, at least without higher-order wizardry (less so with ε-TEX). It is also actually impossible

with LuaTEX, for the reasons mentioned in the first paragraph of this section: commented material has already been discarded, multiple spaces have been reduced, etc. However, for short snippets, our pseudo-verbatim will be quite useful and easy. Let's restate the problem. Suppose we want to be able to write something like:

```
... some fascinating code%
\footnote*{That is \verb"\def\luatex{Lua\TeX}".}
```

i.e. we want verbatim code to appear in a footnote. This can't be done by traditional means, because `\footnote` scans its argument, including the code, and fixes catcodes; hence `\def` is a control sequence and cannot be turned back to four characters. The code below doesn't change that state of affairs; instead it examines and manipulates tokens in the `token_filter` callback. Here's the TEX side (which uses `"..."` instead of the more verbose `\verb"..."`); it simply opens a group, switches to a typewriter font, and registers our Lua function in the callback:

```
\catcode`\"=13
\def"{\bgroup\tt
  \directlua{callback.register("token_filter",
                               verbatim)}%
}
```

And now the Lua side:

```
function verbatim ()
  local t = token.get_next()
  if t[1] > 0 and t[1] < 13 then
    if t[2] == 34 then
      callback.register("token_filter", nil)
      return token.create("egroup")
    else
      local cat = (t[2] == 32 and 10 or 12)
      return {cat, t[2], t[3]}
    end
  else
    return {token.create("string"), t}
  end
end
```

It reads as follows: first we fetch the next token. If it isn't a command, i.e. if its command code is between 1 and 12, then it may be the closing double quote, with character code 34; in this case, we unregister the function and pass to TEX a token created on the fly with `token.create`, a function that produces a token from (among others) a string: here we simply generate `\egroup`. If the character isn't a double quote, we return it but change its command code (i.e. its catcode) to 12 (or 10 if it is a space), thus turning specials to simple characters (letters also lose their original catcode, but that is harmless). We return our token as a table with the three entries mentioned above for the token representation. Finally, if the token is a command, we return a table representing a list of tokens which TEX will read one after the other: the first is `\string`, the second is the original token.

If the reader experiments with the code, s/he might discover that the double quote is actually seen twice: first, when it is active (hence, a command), and prefixed with `\string`; then as the result of the latter operation. Only then does it shut off the processing of tokens.

Inserting discretionaries

Now TEX has read and interpreted tokens. Among the things which have happened, we will now be interested in the following: the nodes that TEX has created and concatenated into a horizontal list. This is where typesetting proper begins. The `hyphenate` callback receives the list of nodes that is the raw material with which the paragraph will be built; it is meant to insert hyphenation points, which it does by default if no function is registered.

In this callback and others, it is instructive to know what nodes are passed, so here's a convenient function that takes a list of nodes and prints their `id` fields to the terminal and log (what number denotes what type of node is explained in chapter 8 of the LuaTEX reference manual), unless the node is a glyph node (`id` 37, but better to get the right number with `node.id`), in which case it directly prints the character:

```
local GLYF = node.id("glyph")
function show_nodes (head)
  local nodes = ""
  for item in node.traverse(head) do
    local i = item.id
    if i == GLYF then
      i = unicode.utf8.char(item.char)
    end
    nodes = nodes .. i .. " "
  end
  texio.write_nl(nodes)
end
```

Let's register it at once in the `hyphenate` callback:

```
callback.register("hyphenate", show_nodes)
```

No hyphenation point will be inserted for the moment, we'll take care of that later.

Now suppose we're at the beginning of some kind of postmodern minimalist novel. It starts with a terse paragraph containing exactly two words:

Your office.

What list of nodes does the `hyphenate` callback receive? Our `show_nodes` function tells us:

50 8 0 Y o u r 10 O f f i c e . 10

First comes a `temp` node; it is there for technical reasons and is of little interest. The node with `id` 8 is a whatsit, and if we asked we'd learn its subtype is 6, so it is a `local_par` whatsit and contains, among other things, the paragraph's direction of writing. The third node is a horizontal list, i.e. an hbox; its subtype (3) indicates that it is the indentation box, and if we queried its width we would be returned the value of `\parindent` (when the paragraph was started) in scaled points (to be divided by 65,536 to yield a value in TEX points).

The nodes representing characters have many fields, among them `char` (a number), which our `show_nodes` function uses to print something a little more telling than an `id` number, `width`, `height` and `depth` (numbers too, expressing dimensions in scaled points), and `font` (yet another number: fonts are internally represented by numbers). Their `subtype` field will be of interest later.

Finally, the nodes with `id` 10 are glues, i.e. the space between the two words and the space that comes from the paragraph's end of line (which wouldn't be there if the last character was immediately followed by `\par` or a comment sign). Their specifications can be accessed via subfields to their `spec` fields (because a glue's specs constitute a node by themselves).

Now, what can be done in this callback? Well, first and foremost, insert hyphenation points into our list of nodes as LuaTEX would have done by itself, had we left the callback empty. The `lang.hyphenate` function does this:

```
callback.register("hyphenate",
  function (head, tail)
    lang.hyphenate(head)
    show_nodes(head)
  end)
```

There is no need to return the list, because LuaTEX takes care of it in this callback, as is also the case with the `ligaturing` and `kerning` callbacks. Also, those three callbacks take two arguments: `head` and `tail`, respectively the first and last nodes of the list to be processed. The tail can generally be ignored.

Now we we can see what hyphenation produces:

50 8 0 Y o u r 10 o f 7 f i c e . 10

As expected, a discretionary has been inserted with `id` 7; it is a discretionary node, with `pre`, `post` and `replace` fields, which are equivalent to the first, second and third arguments of a `\discretionary` command: the `pre` is the list of nodes to be inserted before the line break, the `post` is the list of nodes to be inserted after the line break, and the `replace` is the list of nodes to be inserted if the hyphenation point isn't chosen. In our case, the `pre` field contains a list with only one node, a hyphen character, and the other fields are empty.

A final word on hyphenation. The exceptions loaded in `\hyphenation` can now contain the equivalent of `\discretionary`, by inserting `{pre}{post}{replace}` sequences; German users (and probably users of many other languages) will be delighted to know that they no longer need to take special care of *backen* in their document; a declaration such as the following suffices:

```
\hyphenation{ba{k-}{}{c}ken}
```

Also, with a hyphen as the first and third arguments, compound words can be hyphenated properly.

Ligatures

As its name indicates, the `ligaturing` callback is supposed to insert ligatures (this happens by itself if no function is registered). If we used the `show_nodes` function here, we'd see no difference from the latest output, because that callback immediately follows `hyphenate`. But we can register our function after ligatures have been inserted with the `node.ligaturing` function (again, no return value):

```
callback.register("ligaturing",
  function (head, tail)
    node.ligaturing(head)
    show_nodes(head)
  end)
```

And this returns:

50 8 0 Y o u r 10 o 7 c e . 10

Did something go wrong? Why is *office* thus mangled? Simply because there is an interaction between hyphenation and ligaturing. If the hyphenation point is chosen, then the result is `of-<fi>ce`, where `<fi>` represents a ligature; if the hyphenation point isn't chosen, then we end up with `o<ffi>ce`, i.e. another ligature; in other words, what ligature is chosen depends on hyphenation. Thus the discretionary node has `f-` in its `pre` field, `<fi>` in `post` and `<ffi>` in `replace`.

Ligature nodes are glyph nodes with `subtype` 2, whereas normal glyphs have `subtype` 1; as such, they have a special field, `components`, which points to a node list made of the individual glyphs that make up the ligature. For instance, the components of an `<ffi>` ligature are `<ff>` and `i`, and the components of `<ff>` are `f` and `f`. Ligatures can thus be decomposed when necessary.

How does LuaTEX (either as the default behavior of the `ligaturing` callback or as the

`node.ligaturing` function) know what sequence of glyph nodes should result in a ligature? The information is encoded in the font: LuaTEX looks up the `ligatures` table associated (if any) with each character, and if the following character is included in that table, then a ligature is created. For instance, for `f` in Computer Modern Roman, the `ligatures` table has a cell at index 105, that is `i`, which points to character 12, which contains the `<fi>` ligature. Thus, LuaTEX knows nothing about ligatures involving more than two glyphs. Even the `<ffi>` ligature is a ligature between `<ff>` and `i`.

However, fonts, especially of the OpenType breed, sometimes define ligatures with more than two glyphs; for instance, the input `3/4` is supposed to produce something like ¾ (a single glyph). One can choose, when creating the font from the OpenType file, to create a phantom ligature `<3/>` and make ¾ a ligature between `<3/>` and `4`; then LuaTEX can handle it automatically. It is more elegant and less error-prone, though, to deal with such ligatures by hand, so to speak: register a function in the `ligaturing` callback which, given a string of nodes, creates a ligature. It is also slower.

Also in this callback, such things as contextual substitutions should take place. For instance, initial and final forms of a glyph, be it in Arabic or in some flourished Latin font, should be handled here. In theory, that is quite easy: check the context of a node, i.e. the surrounding nodes; if it matches the context for a given substitution, then apply it. For instance, if our example paragraph were typeset in Minion (shipped gratis with Adobe Reader) with the `ss02` feature on, the *r* of *Your* and the *e* of *office* would be replaced by their final variants, because the contexts match: *r* is followed by a glue and *e* is followed by a stop (technically, they're not followed by glyphs inhibiting the substitution, that is, glyphs denoting a letter). In practice, however, things are more complicated, if only because you have to read such contextual substitutions from the font file.

However, we can perform a very simple type of contextual substitution. Code used to load a font in LuaTEX generally applies the `trep` feature (inspired by XeTEX), so that the grave and single quote characters are replaced with left and right quotes; but one might want to be lazier still and use `"` everywhere; then the proper quotation mark should be substituted, depending on where the double quote occur.

Here's some simple code to implement this rule for such substitutions: if `"` is found, replace it with ' if the node immediately preceding (if any) is a glyph and its character isn't a left parenthesis; otherwise, replace it with ‘. (Here and elsewhere, I use `not (x == y)` where `x ~= y` would be simpler, but `~` would be expanded in `\directlua`, and `x \noexpand~= y` isn't so simple anymore.)

```
local GLYF = node.id("glyph")
callback.register("ligaturing",
  function (head)
    for glyph in node.traverse_id(GLYF, head) do
      if glyph.char == 34 then
        if glyph.prev and glyph.prev.id == GLYF
           and not (glyph.prev.char == 40) then
          glyph.char = 39
        else
          glyph.char = 96
        end
      end
    end
    node.ligaturing(head)
  end)
```

Note that we still apply `node.ligaturing`. Now one can use `"word"` to print ‘word’ and thus rediscover the thrill of modern word processors.

Inserting kerns

Analogous to `ligaturing`, `kerning` is supposed to insert font kerns, and again this happens by itself if no function is registered. Furthermore, nothing happens between the `ligaturing` and `kerning` callbacks, so again, using `show_nodes` would be uninformative. The equivalent function in this case is `node.kerning`, so we can do:

```
callback.register("kerning",
  function (head, tail)
    node.kerning(head)
    show_nodes(head)
  end)
```

And the result is:

50 8 0 Y 11 o u r 10 o 7 c e . 10

What has changed is that a node of `id` 11, i.e. a kern, has been inserted, because an *o* after a *Y* looks better if it is moved closer to it. Compare ‘Yo’ and ‘Yo’. Such kerns are specified in the fonts, like ligatures, which make sense, since kerns, like ligatures, depends on the glyphs' design.

Like contextual ligatures and substitutions, there is contextual positioning. Kerns are encoded in the font (as internalized by LuaTEX) like ligatures, i.e. glyphs have a `kerns` table indexed with character numbers and dimensions (in scaled points) as values; hence kerning is automatic with glyph pairs only, and contextual positioning should be made by hand. For instance, in ‘A.V.’, a (negative) kern should be inserted between the first stop and

the V; however, this should happen only when the stop is preceded by an A; if the first letter were T (i.e. 'T.V.'), the kern is much less desirable (or at least a different amount of kerning should be used).

Finally, some hand-made kerning is to take place here too. For instance, French typographic rules require that a thin space be inserted before some punctuation marks (not so thin, actually, which sometimes verges on the ugly, but one is free to adopt Robert Bringhurst's 'Channel Island compromise'), but there are some exceptions: for instance, although a question mark should be preceded by such a space, the rule obviously doesn't apply if the mark follows an opening parenthesis (?) or another question mark. Technically, this could be encoded in the font; in practice, it is much easier to handle in the `kerning` callback. If one chooses to do so, one should make sure all the newly inserted kerns are of `subtype` 1, because kerns of `subtype` 0 are font kerns and might be reset if the paragraph is built with font expansion.

One last callback before building the paragraph

The previous callbacks apply no matter whether we're building a paragraph or creating an `\hbox`. The ones we'll see now are used only in the first case, i.e. (TEXnically speaking) when a vertical command is encountered in unrestricted horizontal mode. The first of those is `pre_linebreak_filter`, and if we use `show_nodes` on the list it is given, then we notice that something has happened between the `kerning` callback and now:

8 0 Y 11 o u r 10 o 7 c e . 12 10

First, the temporary node at the beginning of the list has been removed; it is not needed anymore, but from now on we should always return the list. Hence, the `show_nodes` call would have been embedded in:

```
callback.register("pre_linebreak_filter",
  function (head)
    show_nodes(head)
    return head
  end)
```

Second, a new node has been inserted, with `id` 12: that is a penalty. If we queried its `penalty` field, it'd return 10,000. Where does the infinite penalty come from? The reader might know that, when preparing to build a paragraph, TEX removes a last space (i.e. the last glue node) of the horizontal list and replaces it with a glue whose value is `\parfillskip`, and prefixes the latter with an infinite penalty so no line break can occur. That is what has happened here: the last node is a glue (`id` 10), but not the same as before, as its `subtype` (15) would indicate: it is the `\parfillskip` glue.

Nothing in particular is supposed to happen in the `pre_linebreak_filter` callback, and TEX does nothing by default. The callback is used for special effects before the list is broken into lines; its arguments are the head of the list to be processed and a string indicating in which circumstances the paragraph is being built; relevant values for the latter are an empty string (we're in the main vertical list), `vbox`, `vtop` and `insert`.

At last, building the paragraph!

Finally we must build the paragraph. To do so we use the `linebreak_filter` callback; by default, paragraph building is automatic (fortunately), but if we register a function in the callback we should break lines by ourselves. Well, more or less: as usual, there is a function, `tex.linebreak`, which does exactly that

The callback receives two arguments: a list of nodes and a boolean; the latter is `true` if we're building the part of a larger paragraph before a math display, otherwise it is false. Now, given the list of nodes, one must return another list of an entirely different nature: it should be made of horizontal boxes (lines of text), glues (interline glues), penalties (e.g. widow and club penalties), perhaps inserts or `\vadjust`-ed material, etc. As just mentioned, the `tex.linebreak` function does all this; it can also take an optional argument, a table with TEX parameters as keys (for instance `hsize`, `tolerance`, `widowpenalty`), so paragraphs can easily be built with special values.

As an example of paragraph building, let's address the issue of setting a paragraph's first line in small caps, as is often done for the first paragraph of a chapter. We're using LuaTEX, so we don't want any dirty trick, and we want TEX to build the best paragraph (i.e. we don't want to simply mess with space in the first line), which includes the possibility that the first line is hyphenated. The code below is just a sketch, but it gives an overview of the approach. First things first, we need a font, and we need a macro to declare that the next paragraph should be treated in a special way:

```
\font\firstlinefont=cmcsc10
\def\firstparagraph{\directlua{
 callback.register("hyphenate", false)
 callback.register("ligaturing", false)
 callback.register("kerning", false)
```

```
callback.register("linebreak_filter",
 function (head, is_display)
   local par, prevdepth, prevgraf =
     check_par(head)
   tex.nest[tex.nest.ptr].prevdepth=prevdepth
   tex.nest[tex.nest.ptr].prevgraf=prevgraf
   callback.register("hyphenate", nil)
   callback.register("ligaturing", nil)
   callback.register("kerning", nil)
   callback.register("linebreak_filter", nil)
   return par
 end)}}
```

First, we deactivate the first three node-processing callbacks by registering `false`, because we want to keep the original list of nodes with only the replacements that occur before the `pre_linebreak_filter` callback; we'll do hyphenating, ligaturing and kerning by hand. In practice, it would be preferable to retrieve the functions (if any) that might be registered in those callbacks and use those, because there might be more than what is done by default. We could do that with `callback.find` but won't bother here.

Next, we register `linebreak_filter` function that calls `check_par`; the latter will return a paragraph and the new value for `prevdepth` and `prevgraf`, so we can set the values for the current nesting level (the list we're in) by hand (it isn't done automatically). Finally, we return the callbacks to their default behavior by registering `nil`.

Before turning to the main `check_par` function, here's a subfunction that it uses to do the job we've prevented LuaTEX from doing by itself: insert hyphenation points, ligatures and kerns, and then build the paragraph. There's no need to set `head` to the return value of `lang.hyphenate`, since no new head can be produced (no hyphenation point can be inserted at the beginning of the list), and anyway `lang.hyphenate` returns a boolean indicating success or failure. Besides the paragraph itself, `tex.linebreak` also returns a table with the values of `prevdepth` and `prevgraf` (and also `looseness` and `demerits`). The last line of the code retrieves the inner numerical representation of the font we've chosen for the first line.

```
local function do_par (head)
  lang.hyphenate(head)
  head = node.ligaturing(head)
  head = node.kerning(head)
  local p, i = tex.linebreak(head)
  return p, i.prevdepth, i.prevgraf
end
local firstlinefont = font.id("firstlinefont")
```

Now we can turn to the big one, called by `linebreak_filter`. First, it builds a tentative paragraph; it works on a copy of the original list because we don't want to spoil it with hyphenation points that might be removed later. Then it finds the first line of the paragraph (the head of the paragraph list might be a glue, or `\vadjust-pre`'d material).

```
local HLIST = node.id("hlist")
local GLYF  = node.id("glyph")
local KERN  = node.id("kern")
function check_par (head)
  local par =  node.copy_list(head)
  par, prevdepth, prevgraf = do_par(par)
  local line = par
  while not (line.id == HLIST) do
    line = line.next
  end
```

Next, in that first line, we check whether all glyphs have the right font; as soon as we find one which isn't typeset in small caps (our `firstlinefont`), we review all the glyphs in the original list until we find the first one that isn't typeset in small caps, and we change its font as we want it. The reader can perhaps see where this is headed: we'll rebuild the paragraph as often as necessary, each time turning one more glyph of the original horizontal list to a small capital, until all the glyphs in the first line are small caps; that is also why we must reinsert hyphenation points, ligatures and kerns each time: fonts have changed, so the typesetting process must be relaunched from the start.*

```
  local again
  for item in node.traverse_id(GLYF, line.head)
  do if not (item.font == firstlinefont) then
    again = true
    for glyph in node.traverse_id(GLYF, head)
    do if not (glyph.font == firstlinefont) then
      glyph.font = firstlinefont
      break
    end; end
    break
  end; end
```

If we must typeset `again`, free the paragraph from TEX's memory and start again with the modified head:

```
  if again then
    node.flush_list(par)
    return check_par(head)
```

* The user might wonder what `line.head` stands for in the second line; that is the same thing as `line.list`, i.e. it gets the contents of a list (its first node). Since LuaTEX v.0.65, `list` has been replaced with `head` for reasons not so clearly explained in my previous paper (see *TUGboat* 31:3); `list` should remain (deprecated) until around v.0.8.

Otherwise (our first line is good, all glyphs are small caps), there's one more thing to check; suppose the last character we turned to small capital was x. By definition, x is at the end of the first line before its font is changed; but is it still the case after the change? Not necessarily: TEX may very well have decided that, given x's new dimensions, it should be better to break before — and perhaps not immediately before x but a couple glyphs before. So perhaps we ended up with small capitals in the second line. They must be removed, but how? Turn them back to lowercase and build the paragraph again? No, definitely not, we'd be stuck in a loop (lowercase in the first line, small caps in the second line, and again ...). The solution adopted here is to turn those glyphs to the original font (say `\tenrm`) and keep them where they are:

```
  else
    local secondline = line.next
    while secondline
          and not (secondline.id == HLIST) do
      secondline = secondline.next
    end
    if secondline then
      local list = secondline.head
      for item in node.traverse_id(GLYF,list)
      do if item.font == firstlinefont then
          item.font = font.id("tenrm")
        else
          break
      end; end
```

Now, what if those first glyphs in the second line were f and i; in small caps they presumably did not form a ligature, but now? We should reapply ligatures. And what about kerning? We should remove all font kerns (they have `subtype` 0) and also reapply kerning. Finally we should repack the line to its original width, so that glues are stretched or shrunken to the appropriate values. That is not optimal, but such cases where small caps end up in the second line are very rare.

The last lines delete the original list and return the paragraph with the associated parameters.

```
      list = node.ligaturing(list)
      for kern in node.traverse_id(KERN, list)
      do if kern.subtype == 0 then
          node.remove(list, kern)
      end; end
      list = node.kerning(list)
      secondline.head = node.hpack(
             list, secondline.width, "exactly")
    end
    node.flush_list(head)
    return par, prevdepth, prevgraf
  end
end
```

The reader may have spotted more than one flaw in this code. A full solution would have greatly exceeded the limits of this already quite long article. So it is left as an exercise: work out a solution that doesn't rely on the assumption that no functions are registered in the other callbacks, for instance. Or give an alternative way to cope with small capitals in the second line (rebuild the paragraph from that line on?).

Handling the paragraph

The `post_linebreak_filter` callback is very calm after all we've just been through: nothing happens by default. It is passed what `linebreak_filter` returns as its first argument, i.e. a list of horizontal lists, penalties, glues, and perhaps interline material (e.g. inserts). It also receives a second argument, a string as with the `pre_linebreak_filter` callback. In my previous paper, I gave examples of what can be done here, for instance underlining. I won't give another example, but the reader looking for practise could try to adapt to LuaTEX Victor Eijkhout's code in section 5.9.6 of *TEX by Topic*.

The callback should return a paragraph, possibly the same as the one it was passed. That paragraph is then appended to the surrounded vertical list, and what follows is the job of the page builder. Our exploration ends here.

Conclusion

Most of the operations we have reviewed aren't new in TEX: LuaTEX simply gives access to them. Since the very beginning, TEX has read lines and tokens and built lists of nodes (although the hyphenating/ligaturing pass has changed a lot in LuaTEX); that is its job. Control over the typesetting process is what makes TEX so good, perhaps better than any other typography software; LuaTEX brings that control one step further and allows manipulating of the very atoms that make digital typography: characters and glyphs, and a few other technical bells and whistles. In view of the freedom that has been gained, I sometimes tend to find TEX82 and its offspring a bit dictatorial, in retrospect.

◇ Paul Isambert
Université de la Sorbonne Nouvelle
France
zappathustra (at) free dot fr

Luna — my side of the moon

Paweł Jackowski

Perhaps everyone knows the pleasant feeling when a long lasting project is finally done. A few years ago, when I was almost happy with my pdfTEX environment, I saw LuaTEX for the first time. So instead of enjoying some relief, I had to take a deep breath and start moving the world to the Moon. The state of weightlessness thus caused is such that I'm not able to walk on the "normal" ground any more. But I don't even think about going back. Although I still haven't settled for good yet, the adventure is delightful. To domesticate a new environment I gave it a name — Luna.

First thoughts

My first thought after meeting LuaTEX was "wow!". Scripting with a neat programming language, access to TEX lists, an ability to hook some deep mechanisms via callbacks, a font loader library on hand, an integrated METAPOST library and more. All this was tempting and I had no doubts I wanted to go for it. At the first approach I was thinking of migrating my workflows step-by-step, replacing some core mechanisms with those provided by LuaTEX. But these were not only the macros that needed to change. It was considering TEX as a programming language that needed to change. In LuaTEX I rather treat TEX as a paragraph and page building machine to which I can talk in a real programming language.

There were a lot of things I had to face before I was able to typeset anything, beginning with a UTF-8 regime and a new TEX font representation, a lot of work that I never wanted to do myself. So just after "wow!" also "oops..." had to come. In this article I focus on things rather tightly related to PDF graphics, as I find that part the most interesting, at least in the sense of taking advantage of Lua and LuaTEX functionalities.

`\pdfliteral` retires

TEX concentrates on text, providing only a raw mechanism for document graphics features, such as colors, transparencies or geometry transformations. pdfTEX goes a little bit further in providing some concept of a graphic state accessible for the user. But the gear for the graphic control remains the same. We have only specials in several variants.

This article appeared originally in slightly different form in *MAPS* 41, fall 2010.

What's wrong with them? The things which they do behind the scenes may hurt.

```
\def\flip#1{%
  \pdfliteral{q -1 0 0 -1 20 6 cm}%
  \hbox to0pt{#1\hss}%
  \pdfliteral{Q}\hbox to20bp{\hss}}
\def\red#1{%
  \pdfliteral page{q 0 1 1 0 k}%
  #1\pdfliteral page{Q}}
```

The first macro applies a transformation to a ʇxǝʇ object, the second applies a color (red online, gray in print). If used separately, they work just fine. If used as `\flip{\red{text}}`, it's still ok: ʇxǝʇ. Now try to say `\red{\flip{text}}`. The text is transformed and colored as expected. But all the rest of the page is broken, as its content is completely displaced! And now try `\red{\flip{text}?}` (with a question mark at the end of a parameter text). Everything is perfectly ok again: ʇxǝʇ?

Here is what happens: when `\pdfliteral` occurs, pdfTEX inserts a whatsit. This whatsit will cause writing the data into the output PDF content stream at the shipout time. If the literal was used in a default mode (with no `direct` or `page` keywords) pdfTEX first writes a transformation from lower-left corner of the page to the current position, then prints the user data, then writes another transformation from the current position back to the PDF page origin. Actually the transform restoration is not performed immediately after writing the user data, but at the beginning of the very next textual node. So in the case of several subsequent literal whatsit nodes, the transform may not occur where the naive user expects it. Simplifying the actual PDF output, we expected something like

```
q 0 1 1 0 k          % save, set color
1 0 0 1 80 750 cm    % shift to TeX pos
q -1 0 0 -1 20 6 cm  % save, transform
BT ... ET            % put text
Q                    % restore transform
1 0 0 1 -80 -750 cm  % shift (redundant)
Q                    % restore color
```

but we got

```
q 0 1 1 0 k
1 0 0 1 80 750 cm
q -1 0 0 -1 20 6 cm
BT ... ET
Q
Q
1 0 0 1 -80 -750 cm
```

In general, the behavior of \pdfliterals depends on the surrounding node list. There are reasons behind it. Nevertheless, one can hardly control lists in pdfTEX, so it's hard to avoid surprises.

Does LuaTEX provide something better then \pdfliterals? Yes; it provides \latelua. Very much like \pdfliteral, a \latelua instruction inserts a whatsit. At shipout time, LuaTEX executes the Lua code provided as an argument to \latelua. The code may call the standard pdf.print() function, which writes raw data into a PDF content stream. So, what's the difference? The difference is that in \latelua chunks we know the current position on the page: it is accessible through the pdf.h and pdf.v fields. We can use the position coordinates explicitly in the literal content. To simulate the behavior of \pdfliteral one can say

```
\latelua{
  local bp = 65781
  local cm = function(x, y)
    return string.format(
     "1 0 0 1 \%.4f \%.4f cm\string\n",
     x/bp, y/bp
    )
  end
  pdf.print("page", cm(pdf.h, pdf.v))
  % special contents
  pdf.print("page", cm(-pdf.h, -pdf.v))
}
```

now having the \latelua mechanism and the pdf.print() function, I no longer need and no longer use \pdfliteral.

Graphic state

Obviously writing raw PDF data is supposed to be covered by lower level functions. Here is an example of how I set up graphic features in a higher level interface:

```
\pdfstate{
  local cmyk = color.cmyk
  cmyk.orange =
    (0.8*cmyk.red+cmyk.yellow)/2
  fillcolor = cs.orange
  opacity = 30
  linewidth = '1.5pt'
  rotate(30)
  ...
}
```

The definition of \pdfstate is something like

```
\long\def\pdfstate#1{%
  \latelua{setfenv(1, pdf) #1}}
```

The parameter text is Lua code. The setfenv() call simply allows me to omit the 'pdf.' prefix before variables. Without that I would need

```
\latelua{
  pdf.fillcolor = pdf.color.cmyk.orange
  pdf.opacity = 30
  pdf.linewidth = '1.5pt'
  pdf.rotate(30)
  ...
}
```

pdf is a standard LuaTEX library. I extend its functionality, so every access to special fields causes an associated function call. Each such function updates the internal representation of a graphic state and keeps the output PDF graphic state synchronized by writing appropriate content stream data. But whatever happens underneath, on top I have just key=value pairs. I'm glad I no longer need to think about obscure TEX interfaces for that. The Lua language is the interface.

I expect graphic features to behave more or less like basic text properties, a font selection and size. They should obey grouping and they should be passed through page breaks. The first requirement can be simply satisfied by \aftergroup in conjunction with \currentgrouplevel. A simple group-respecting graphic state could be made as the following:

```
\newcount\gstatelevel
\def\pdfsave{\latelua{
  pdf.print("page", "q\string\n")}}
\def\pdfrestore{\latelua{
  pdf.print("page", "Q\string\n")}}
\def\pdflocal#1{
  \ifnum\currentgrouplevel=\gstatelevel
  \else
    \gstatelevel=\currentgrouplevel
    \pdfsave \aftergroup\pdfrestore
  \fi \latelua{pdf.print"#1\string\n"}}

\begingroup \pdflocal{0.5 g}
this is gray
\endgroup
this is black
```

Passing a graphic state through page breaks is relatively difficult due to the fact that we usually don't know where TEX thinks the best place to break is. In my earth-life I abused marks for that purpose or, when a more robust mechanism was needed, I used \writes at the price of another TEX run and auxiliary file analysis. And so we come to another advantage of using \latelua. Recalling the fact that Lua chunks are executed during shipout, we

don't need to worry about the page break because it has already happened. If every graphic state setup is a Lua statement performed in order during shipout and every such statement keeps the output PDF state in sync through `pdf.print()` calls, then after the shipout the graphic state is what should be passed to the very next page.

In a well-structured PDF document every page should refer only to those resources which were actually used on that page. The pdfTEX engine guarantees that for fonts and images, while the `\latelua` mechanism makes it straightforward for other resource types.

Note a little drawback of this late graphic state concept: before shipout one can only access the state of the beginning of the page, because recent `\latelua` calls that will update the current state have not happened yet. I thought this might be a problem and made a mechanism that updates a pending-graphic state for early usage, but so far I have never needed to use it in practice.

PDF data structures

When digging deeper, we have to face creating custom PDF objects for various purposes. Due to the lack of composite data structures, in pdfTEX one was condemned to using strings. Here is an example of PDF object creation in pdfTEX.

```
\immediate\pdfobj{<<
/FunctionType 2
/Range [0 1 0 1 0 1 0 1]
/Domain [0 1] /N 1
/C0 [0 0 0 0] /C1 [0 .4 1 0]
>>}
\pdfobj{
  [/Separation /Spot /DeviceCMYK
  \the\pdflastobj\space 0 R]
}\pdfrefobj\pdflastobj
```

In LuaTEX one can use Lua structures to represent PDF structures. Although it involves some heuristics, I find it convenient to build PDF objects from clean Lua types, as in this example:

```
\pdfstate{create
  {"Separation","Spot","DeviceCMYK",
    dict.ref{
      FunctionType = 2,
      Range = {0,1,0,1,0,1,0,1},
      Domain = {0,1}, N = 1,
      C0 = {0,0,0,0}, C1 = {0,.4,1,0}
    }
  }
}
```

Usually, I don't need to create an independent representation of a PDF object in Lua. I rather operate on more abstract constructs, which may have a PDF-independent implementation and may work completely outside of LuaTEX. For a color representation and transformations I use my color library, which has no idea about PDF. An additional LuaTEX-dependent binding extends that library with extra skills necessary for the PDF graphic subsystem.

Here is an example of a somewhat complex colorspace: a palette of duotone colors, each consisting of two spot components with lab equivalent (the PDF structure representing that is much too long to be shown here):

```
\pdfstate{
  local lab = colorspace.lab{
    reference = "D65"
  }
  local duotone = colorspace.poly{
    {name = "Black", lab.black},
    {name = "Gold",  lab.yellow},
  }
  local palette = colorspace.trans{
    duotone(0,100), duotone(100,0),
    n = 256
  }
  fillcolor = palette(101)
}
```

In the last line, the color object (simple Lua table) is set in a graphic state (Lua dictionary), and its colorspace (another Lua dictionary) is registered in a page resources dictionary (yet another Lua dictionary). The graphic state object takes care to update a PDF content stream and finally the resources dictionary "knows" how to become a PDF dictionary.

It's never too late

When talking about PDF object construction I've concealed one sticky difficulty. If I want to handle graphic setup using `\latelua`, I need to be able to create PDF objects during shipout. Generally, `\latelua` provides no legal mechanism for that. There is the `pdf.obj()` standard function, a LuaTEX equivalent of the `\pdfobj` primitive, but it only obtains an allocated PDF object number. What actually ensures writing the object into the output is a whatsit node inserted by a `\pdfrefobj`⟨*number*⟩ instruction. But in `\latelua` it is too late to use it. We also cannot use the `pdf.immediateobj()` variant within `\latelua`, as

it writes the object into the page content stream resulting in an invalid PDF document.

So what can one do? LuaTEX allows creating an object reference whatsit by hand. If we know the tail of the list currently written out (or any list node not yet swallowed by a shipout procedure), we can create this whatsit and put it into the list on our own (risk), without use of `\pdfrefobj`.

```
\def\shipout{%
  \setbox256=\box\voidb@x
  \afterassignment\doshipout\setbox256=}
\def\doshipout{%
  \ifvoid256 \expandafter\aftergroup \fi
  \lunashipout}
\def\lunashipout{\directlua{
  luna = luna or {}
  luna.tail =
    node.tail(tex.box[256].list)
  tex.shipout(256)
}}

\latelua{
  local data = "<< /The /Object >>"
  local ref = node.new(
    node.id "whatsit",
    node.subtype "pdf_refobj"
  )
  ref.objnum = pdf.obj(data)
  local tail = luna.tail
  tail.next = ref  ref.prev = tail
  luna.tail = ref % for other lateluas
}
```

In this example, before every `\shipout` the very last item of the page list is saved in `luna.tail`. During shipout all code snippets from `late_lua` whatsits may create a `pdf_refobj` node and insert it just after the page tail to ensure writing them out by LuaTEX engine.

Self-conscious `\latelua`

If every `\latelua` chunk may access a page list tail, why not give it access to a `late_lua` whatsit node to which this code is linked? Here is a concept of the whatsit that contains Lua code that can access the whatsit:

```
\def\lateluna#1{\directlua{
  local self = node.new(
    node.id "whatsit",
    node.subtype "late_lua"
  )
  self.data = "\luaescapestring{#1}"
  luna.this = self
  node.write(self)
}}

\lateluna{print(luna.this.data)}
```

Beyond the page builder

Self-printing Lua code is obviously not what I use this mechanism for. It is worthy to note that if we can make a self-aware `late_lua` whatsit, we can access the list following this whatsit. It is too late to change previous nodes, as they were already eaten by a shipout and written to the output, but one can freely (which doesn't mean safely!) modify nodes that follow the whatsit.

Let's start with a more general self-conscious `late_lua` whatsit:

```
\long\def\lateluna#1{\directlua{
  node.write(
    luna.node("\luaescapestring{#1}")
  )
}}
\directlua{
luna.node = function(data)
  local self = node.new(
    node.id "whatsit",
    node.subtype "late_lua"
  )
  local n = \string#luna+1
  luna[n] = self
  self.data =
    "luna.this = luna["..n.."] "..data
  return self
end
}
```

Here is a function that takes a text string, font identifier and absolute position as arguments and returns a horizontal list of glyph nodes:

```
local string = unicode.utf8
function luna.text(s, font_id, x, y)
  local head = node.new(node.id "glyph")
  head.char = string.byte(s, 1)
  head.font = font_id
  head.xoffset = -pdf.h+tex.sp(x)
  head.yoffset = -pdf.v+tex.sp(y)
  local this, that = head
  for i=2, string.len(s) do
    that = node.copy(this)
    that.char = string.byte(s, i)
    this.next = that  that.prev = this
    this = that
  end
  head = node.hpack(head)
```

```
  head.width = 0
  head.height = 0
  head.depth = 0
  return head
end
```

Now we can typeset texts even during shipout. The code below results in typing `it is never too late!` text with 10bp offset from the page origin.

```
\lateluna{
  local this = luna.this
  local text = luna.text(
    "it is never too late!",
    font.current(), '10bp', '10bp'
  )
  local next = this.next
  this.next = text  text.prev = this
  if next then
    text = node.tail(text)
    text.next = next  next.prev = text
  end
}
```

Note that when mixing shipout-time typesetting (manually generated lists) and graphic state setups (using `pdf.print()` calls), one has to ensure placing things in order. Once a list of glyphs is inserted after a `late_lua` whatsit, the embedded Lua code should not print literals into the output. All literals will effectively be placed before the text anyway. Here is a funny mechanism to cope with that:

```
\lateluna{
luna.thread = coroutine.create(
function()
  local this, next, text, tail
  for i=0, 360, 10 do
    % graphic setup
    pdf.fillcolor =
      pdf.color.hsb(i,100,100)
    pdf.rotate(10)
    % glyphs list
    this = luna.this  next = this.next
    text = luna.text("!",
      font.current(), 0, 0)
    this.next = text  text.prev = this
    text = node.tail(text)
    % luna tail
    tail = luna.node
      "coroutine.resume(luna.thread)"
    text.next = tail tail.prev = text
    if next then
      tail.next = next next.prev = tail
    end
    coroutine.yield()
  end
end)
coroutine.resume(luna.thread)
}\end
```

This is the output (regrettably grayscaled in print):

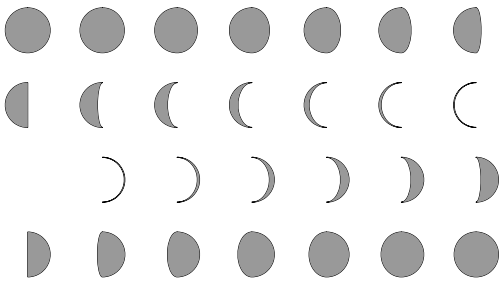
Once the page shipout starts, the list is almost empty. It contains just a `late_lua` whatsit node. The code of this whatsit creates a Lua coroutine that repeatedly sets some color, some transformation and generates some text (an exclamation mark) using the method previously shown. A tail of the text is another `late_lua` node. After inserting the newly created list fragment, the thread function yields, effectively finishing the execution of the first `late_lua` chunk. Then the shipout procedure swallows the recently generated portion of text, writes it out and takes care of font embedding. After the glyph list the shipout spots the `late_lua` whatsit with the code that resumes the thread and performs another loop iteration, making a graphic setup and generating text again. So the execution of the coroutine starts in one whatsit, but ends in another, which didn't exist when the procedure started. Every list item is created just before being processed by the shipout.

Reinventing the wheel

Have you ever tried to draw a circle or ellipse using `\pdfliterals`? It is very inconvenient, because PDF format provides no programming facilities and painting operations are rather limited in comparison with its PostScript ancestors. Here is an example of PostScript code and its output. The code uses control structures, which are not available in PDF. It also takes an advantage of the `arc` operator that approximates arcs with Bézier curves. To obtain elliptical arcs, it uses the fact that (unlike in PDF) transformations can be applied between path construction operators.

```
/r 15 def
/dx 50 def /dy -50 def
/pos {day 7 mod dx mul week dy mul} def
/arx /arc load def

dx dy 4 mul neg translate
0.6 setgray 0.4 setlinewidth
1 setlinejoin 1 setlinecap
0 1 27 {
  /day exch def /week day 7 idiv def
  /s day 360 mul 28 div cos def
  day 14 eq {
    /arx /arcn load def
  } {
    gsave pos r 90 270 arx
    day 7 eq day 21 eq or {
      closepath
      gsave 0 setgray stroke grestore
    } {
      s 1 scale
      pos exch s div exch r 270 90 arx
      gsave 0 setgray initmatrix stroke
      grestore
    } ifelse
    fill grestore
  } ifelse
} for
```



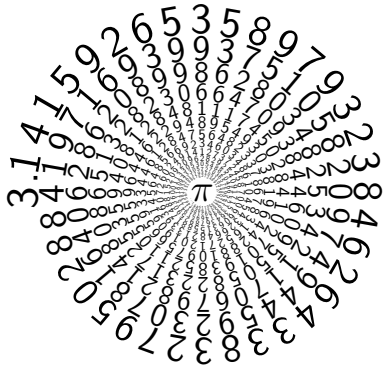
In LuaTEX one can hire METAPOST for drawings, obtaining a lot of coding convenience. The above program wouldn't be much simpler, though. As of now, METAPOST does not generate a PDF; whatever it outputs still needs some postprocessing to include the graphic on-the-fly in the main PDF document.

As I didn't want to invent a completely new interface for graphics, I decided to involve PostScript code in document creation. Just to explain how it may pay off, after translating the example above into a PDF content stream I obtain 30k bytes of code, which is quite a lot in comparison with the 500 bytes of PostScript input.

PostScript support sounds scary. Obviously I'm not aiming to develop a fully featured PostScript machine on the LuaTEX platform. A PostScript interpreter is supposed to render the page on the output. In Luna I just write a vector data string into a PDF document contents, so what I actually need is a reasonable subset of PostScript operators. The aim is to control my document graphics with a mature language dedicated for that purpose. The following two setups are equivalent, as at the core level they both operate on the same Lua representation of a graphic state.

```
\pdfstate{% lua interface
  save()
  fillcolor = color.cmyk(0,40,100,0)
  ...
  restore()}
\pdfstate{% postscript interface
  ps "gsave 0 .4 1 0 setcmykcolor"
  ...
  ps "grestore"
}
```

A very nice example of the benefit from joining typesetting beyond the page builder and PostScript language support is this π-spiral submitted by Kees van der Laan:



(See http://gust.org.pl/projects/pearls/2010p.)

◇ Paweł Jackowski
GUST

Reflections on the history of the LaTeX Project Public License (LPPL) — A software license for LaTeX and more

Frank Mittelbach

Abstract

In August 2010 the LaTeX Project Public License (LPPL) was finally listed by the Open Source Initiative (OSI) as a free software license. This marks the endpoint of a long set of discussions around the TeX community's predominant license.

This article reflects on the history of the license, the way it came about, and the reasons for its development and content. It explains why it was chosen even though alternative free licenses have been available at least from 1990 onwards.

Contents

1 Introduction: Why a special license?

1.1 The early days of the TeX community

When Donald Knuth in 1982 published the TeX program it was one of the first, if not indeed the first, major program to be published as documented source in its entirety. As such it forms an early example of "free software", well before this term was more formally defined by Richard Stallman in the free software movement, and well before one of the most influential free software licenses — the GNU GPL — was released in 1989.

Instead of a more elaborate license the source of the TeX program contained (and still contains) the interesting copyright and license information shown in Figure 1 on the following page. The motivation for this small set of clauses was to ensure that documents written for the TeX engine would be readable for the foreseeable future and indeed, TeX and its extensions still compile documents written in the early 1980s and produce output exactly as intended.

In those days, when the TeX community was born, the Internet was mainly restricted to academics and used for knowledge sharing. Commercial aspects hadn't yet entered the space and spam wasn't a known phenomenon. As a result, not much got formalized and there was a general trust that others would respect your ideas and would together help in improving and developing them. People spent countless hours on developing code and ideas and made them available (typically free of charge) to others. Most people didn't bother with any formal copyright notice, some had statements like "Copyright *date name* All rights reserved" while others explicitly placed their work in the "public domain".

Legally, all such works that were developed without granting explicit rights to others (e.g., by stating those rights in a license, or by placing the material into the public domain), didn't offer anybody a right to work with the material, or to use it for private or commercial purposes without explicitly obtaining this right from the copyright holder. So the simple copyright line "`Copyright (C) 1992 by Leslie Lamport`" (from the old LaTeX 2.09 sources) could probably have been used to go after anybody who made use of LaTeX whether for their PhD or for typesetting books for publication and sale.

But of course, nobody understood those lines in this way. They were correctly understood[1] as only marking the intellectual ownership of the code but in the mind of the community and, I would think, in the minds of most if not all of the developers, not as a mechanism to restrict any "proper" use of the material. Now the interesting part here is "proper": as most people spent considerable free time in developing their work, there was a base assumption in the community (and other software communities) that while such work should be freely available, those that use it should also in one way or the other contribute to the whole setup. Commercial use was frowned upon by most as a way to take away the work of others for profit without a benefit for the community, so (not surprisingly) after a while works appeared that explicitly stated "Freely usable for non-commercial usage", or "Commercial use not allowed" in addition to a copyright notice.

Again, I would claim, back then nobody really understood the implications and the legal situation created with such statements — I certainly didn't when I developed my first packages for LaTeX; I

[1] In a legal sense this isn't the correct interpretation as just explained.

```
% This program is copyright (C) 1982 by D. E. Knuth; all rights are reserved.
% Copying of this file is authorized only if (1) you are D. E. Knuth, or if
% (2) you make absolutely no changes to your copy. (The WEB system provides
% for alterations via an auxiliary file; the master file should stay intact.)
% See Appendix H of the WEB manual for hints on how to install this program.
% And see Appendix A of the TRIP manual for details about how to validate it.
```

Figure 1: License of TEX, the program [3]

simply copied such lines that I had seen in other works. Especially a statement like "No commercial use allowed" was way over the top, since everybody was happy if his or her package got used to produce fine books or published articles and in most cases that meant the work was commercially used.

1.2 Digression: The `multicol` license

The fact that such statements were not a dull sword was something I learned to my surprise at one point when I got approached by somebody for special extensions to `multicol` which took me quite some time to implement. At several points in the discussions I asked about the background for the requests and finally got told that they had no intention of telling me or anybody or making any of their part of the work available to others as they wanted to make money from it and that I should stop bothering them. The way this was done got me slightly mad and so I pointed out "heh, have you read the license statement on `multicol` about commercial usage not being allowed?" That made the email correspondence come to an abrupt halt for a moment and then a day or two later I had the company lawyers asking for my phone number in Germany to discuss this and reach some settlement and license agreement. Well, I was certainly young and naive back then[2] so I didn't come out rich from this and probably wouldn't have either way, but it sure felt good that I had a lever to stop being taken for an absolute imbecile that could be made to work for free under false premises.

This was about the first time I got some awareness about the importance and power of licenses as well as of the fact that what was out there wasn't really what people intended. As I wasn't interested in making money from LATEX software and normally would have wanted to use my stuff freely and free of charge, some refinements were really in order. Thus, about to sell my soul and negotiate a special license with this company I had to come up with some idea of an acceptable license (including a license fee). I ended up with a sort of psychological experiment, which was partly my coward's way out of not wanting to deal with license fees and partly some genuine interest on what would happen. The result was perhaps the most curious license ever drawn up in that I required for certain commercial usages of `multicol` the licensee to determine the importance of it for his or her circumstances and determine the license fee from that.

I must say that the experiment as such was a success as it provided me with some interesting insights into human psychology, though I can't recommend it to anybody who wants to make money from software or other works. Not that I want to imply that no license fees got paid: over the years I got a number of nice presents, a book in Japanese (with a 100 Deutschmark note inside I nearly overlooked as it was hidden and nowhere mentioned), and a few other things, so all in all, some pleasant surprises.

Somewhere around 2000 I changed the license for `multicol` to the LPPL but to honor the history (and to continue the experiment) I kept the previous license now toned down to a "Moral Obligation" so people are free to ignore it completely if they wish to, while previously they were only free to set the fee to zero by stating that this is the value they attach to their use of `multicol`.[3]

1.3 The GNU GPL — A new star on the horizon

Returning back to history: in 1989 Richard Stallman published the first version of the GPL (General Public License) [1] for use with programs released as part of the GNU project. Richard intended the GPL to become a license that could be used with any free software project and in that he spectacularly succeeded (Wikipedia reports for 2007 a penetration of roughly 70% on major sites such as `SourceForge.net`). Since its first introduction the use of the GPL in the free software development communities increased steadily to reach these impressive figures, especially in communities that were concerned with developing programs for individual use. The strong copyleft [2] provided by the GPL gave the programmer who used the license the confidence that their work would benefit the whole world and any future development

[2] I can report the first attribute has changed since then.

[3] Interested people can find the wording of this "Moral Obligation" at the top of the `multicol.sty` or `.dtx` file [4]. It is nearly identical to the earlier license statement.

Our aim is that LaTeX should be a system which can be trusted by users of all types to fulfill their needs. Such a system must be stable and well-maintained. This implies that it must be reasonably easy to maintain (otherwise it will simply not get maintained at all). So here is a summary of our basic philosophy:

> *We believe that the freedom to rely on a widely-used standard for document interchange and formatting is as important as the freedom to experiment with the contents of files.*
>
> *We are therefore adopting a policy similar to that which Donald Knuth applies to modifications of the underlying TeX system: that certain files, together with their names, are part of the system and therefore the contents of these files should not be changed unless the following conditions are met:*
>
> - *they are clearly marked as being no longer part of the standard system;*
> - *the name of the file is changed.*

In developing this philosophy, and the consequent limitations on how modifications of the system should be carried out, we were heavily influenced by the following facts concerning the current widespread and wide-ranging uses of the LaTeX system.

1. *LaTeX is not just a document processing system; it also defines a language for document exchange.*
2. *The standard document class files, and some other files, also define a particular formatting of a document.*
3. *The packages that we maintain define a particular document interface and, in some cases, particular formatting of parts of a document.*
4. *The interfaces between different parts of the LaTeX system are very complex and it is therefore very difficult to check that a change to one file does not affect the functionality of both that file and also other parts of the system not obviously connected to the file that has been changed.*

Figure 2: Excerpts from the 1995 document "Modifying LaTeX" [5]

based on their code would remain free, rather than being exploited by software companies that would not return anything back to the community.

Within the TeX — and especially LaTeX — community, however, the GPL played only a niche role.[4] The community starship, the TeX program itself, came with its own very specific license "change my name if you want to change me" and many people (if they had bothered with some explicit license at all) had adopted a similar approach or had used lines like "freely usable for non-commercial purposes" as explained earlier.

1.4 The move from LaTeX 2.09 to LaTeX 2ε

In 1993 the LaTeX project released a fundamentally new version of LaTeX. This new version (LaTeX 2ε) for the first time got an explicit license in the form of a file called `legal.txt` which inside had the title "LaTeX 2ε Copyright, Warranty and Distribution Restrictions". One can think of this file as the very first version of the LPPL, though it wasn't called that in those days and it was a lot simpler than the license under which LaTeX is made available today.

Perhaps the most important aspect of it (which later on also turned out to produce the biggest controversy) was the list of restrictions that apply when producing changed versions of files from the LaTeX system, the most prominent being

> * *You rename the file before you make any changes to it.*

This was directly modeled after Don Knuth's license for TeX and within the TeX community there was broad consensus this this was an adequate approach to balance between the freedom of the individual to be able to reuse and modify the code if so desired and the importance of LaTeX as a communication language where people relied on being able to faithfully reproduce a document written in one place elsewhere.

[4] This situation has changed only marginally over time. The majority of the packages for LaTeX now use the LPPL for their license, though many of the executable support programs and some package use the GPL. More precisely, in October 2010 we had 3849 products/packages listed on CTAN of which 592 (i.e., about 15%) were distributed under GPL and 1751 (i.e., about 45%) used the LPPL; the remainder (many of them fonts) had other licenses. And even if you just look at non-LaTeX works, this means the GPL is used by about 28% so still significantly less than in other free software communities.

This license is an incomplete statement of the distribution terms for LaTeX. As far as it goes, it is a free software license, but incompatible with the GPL because it has many requirements that are not in the GPL.

This license contains complex and annoying restrictions on how to publish a modified version, including one requirement that falls just barely on the good side of the line of what is acceptable: that any modified file must have a new name.

The reason this requirement is acceptable for LaTeX is that TeX has a facility to allow you to map file names, to specify "use file bar when file foo is requested". With this facility, the requirement is merely annoying; without the facility, the same requirement would be a serious obstacle, and we would have to conclude it makes the program non-free.

The LPPL says that some files, in certain versions of LaTeX, may have additional restrictions, which could render them non-free. For this reason, it may take some careful checking to produce a version of LaTeX that is free software.

Figure 3: Excerpts from Richard Stallman's analysis of LPPL 1.2 [6]

1.5 Challenged by GPL evangelists

While the TeX community was content with the status quo, people in the "GPL" world who used TeX and LaTeX felt uncomfortable with the licenses in use and started to lobby for using the GPL within the TeX community, as they felt that it was an unjustified restriction to be forced to change a file name prior to making changes to it. The GPL doesn't pose any such restriction: you can modify a work and distribute it without providing any easy visible clue to its changed content.[5]

In essence two different world views on what is "free software" and who should have what rights clashed head-on for the first time. The GPL view is largely focused on the individual programmer, with the purpose of the GPL being to offer him or her a maximum of rights on using and manipulating the work as well as ensuring that such rights can't subsequently be taken away. On the other hand, the TeX program license and later the LPPL acknowledged the fact that TeX and LaTeX defined a language for communication and that the definition of such a language needs to remain stable to serve as the means of communication, i.e., it tried to achieve a balance between the individual right of a programmer to freely use the work and the community right of the users of this language to rely on the work to be faithfully representing the language itself and thus making communication possible.

In response to suggestions that the modification and distribution conditions for the files constituting the LaTeX system should be similar to those implied by Version 2 of the GPL, the LaTeX project team published the document "Modifying LaTeX" [5] in an attempt to clarify the position of the LaTeX Project team and to explain the rationale behind the license decision. Some relevant excerpts from this document are shown in Figure 2 on the previous page. The document also gave explicit guidance on how to freely change a LaTeX system in any way desired, either through customization or — if really needed — through producing a new system only based on the LaTeX code.

In 1995 Chris Rowley and I also met face to face with Richard Stallman to discuss the Free Software Foundation (FSF) concerns about the LaTeX license and as a result of this meeting and some subsequent email discussions (in which we discussed a number of aspects of the license and clarified or changed several of them), Richard acknowledged LaTeX (and its at that point somewhat incomplete license statement) as free software.[6]

Nevertheless, Richard made it very clear that he didn't like the approach taken by TeX and LaTeX and published his viewpoint as an analysis of the license on the GNU web pages [6] of which excerpts are shown in Figure 3. (The current page states that

[5] This is an oversimplification, as the GPL requires that "You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change." However, in a context like LaTeX where many files are loaded in the background, this would mean a user would need to check hundreds of files for modification information to ensure that he or she is using the official versions when they get loaded by `\usepackage` etc. In contrast, if a file name change in case of modifications is required then this problem vanishes as the documents by themselves identify what they expect.

[6] Looking through my email archives I found the following beauty from the end of this discussion (the slanted text is from Richard, the reply was from some member of the LaTeX project team who shall go unnamed):

> *Ok, I believe that the methods you've described for modifying LaTeX 2e are adequate, so that LaTeX 2e can be considered free software.*
> Hoorah hooray, let the jubilation commence!
> LaTeX is free software after all!

— as it turned out, this conclusion was a bit premature.

this analysis applies to LPPL 1.2, but the excerpt more or less corresponds to the wording that was on the GNU web site at the time.)

I'm not going to attempt to defeat his points here, some are certainly well taken and others are a matter of opinion and anyway, time has moved on because the license text has greatly evolved from the `legal.txt` of 1993, via LPPL 1.0 in early 1999, to LPPL 1.2[7] at the end of 1999, and ultimately to LPPL 1.3 in 2003.

2 The evolution of the LPPL

2.1 An attempted cleanup

In the years between 1995 and 1999 a lot of new software got written for LaTeX and other flavors of TeX. Most of this software was made available through CTAN (the "Comprehensive TeX Archive Network") and regularly through TeX distributions such as "TeX Live" or teTeX. One growing problem faced by these distributions was the number of different licenses under which all such packages in the distribution were made available. LaTeX core had its `legal.txt` but everything else had its own license and though there was the general belief that all or most of it was free software, depending on the viewpoint this wasn't true. In many cases, different strings were attached to individual packages so that a user of, say, LaTeX would have been required to look into every package used to understand what he or she was allowed to do with it.

So in 1998 or thereabouts, an attempt was made to ease this situation: People wanted to produce a "free" TeX distribution for Debian, and to make such a distribution in any way useful it was necessary to ask for license changes in many packages or otherwise exclude them as "non-free". This was quite a heroic act undertaken by a few individuals[8] in their spare time, as it often involved tracking down package authors that had moved on to other pastures and had completely forgotten that they had written some TeX or LaTeX package in their former lives. (According to Karl Berry this kind of work continues to this day, as legal terms unique to a given package continue to be discovered, despite efforts at comprehensive reviews.)

During that time the idea came up to turn LaTeX's `legal.txt` into a generally usable license, so that package authors could simply refer to it as the license they use for their works. So this was the underlying reason for the attempt to write up the LPPL and in March 1999 version 1.0 of this license was made publicly available. But as you can imagine, writing a license is a nontrivial act, thus we had many discussions on how to express the intent behind the license with the result that in fairly rapid succession (all still in 1999) versions 1.1 and 1.2 appeared. Looking back, the changes we made from `legal.txt` via LPPL-1.0 to LPPL-1.1 to LPPL-1.2 were not all necessarily for the better, so some of Richard's criticism certainly holds true for these early attempts. What we changed for LPPL-1.3, with the help of some more legally trained people from Debian, was ultimately considerably more significant.

2.2 Digression: The GNUTeX 0.x project

Concurrently, but unrelated to the efforts mentioned earlier, the GNU project (that was also quite heavily using TeX in the form of Texinfo for documentation) got interested in shipping a "free TeX distribution" with the GNU software and was looking at which of the existing distributions might fit the bill — to which the answer was none.

As Richard later explained to us, he became exceedingly concerned that none of the major TeX distributions could be distributed by GNU so he approached Thomas Esser (maintainer of the teTeX distribution) to ask if it would be possible to separate the distribution into a free and a non-free part. Thomas said yes, but that this would take quite some time. So Richard asked within the GNU project if there would be people to help Thomas with this.

At this point somebody came forward and suggested that he would be interested but would prefer to build a new free distribution from scratch instead of helping to update teTeX. Richard gave his okay and as a result a TeX distribution called GNUTeX got produced. As it turned out (not really a surprise given the license situation in the TeX world) the first public beta of this distribution was very much crippled, but what made matters worse was that it completely ignored the existing license conditions of the packages it included, for example, by not distributing the LaTeX source documentation, i.e., its `.dtx` files.

When this became known to the LaTeX project, David Carlisle tried to reason with the maintainer of this new distribution and asked him to update it. Unfortunately this resulted in very strange statements, such as that he would remove files written by David and have people "write free replacements" for them,

[7] LPPL 1.1 lived for a very short time with only minor differences in wording, but not content, from 1.2 — in essence we had to get our act together and that took some attempts and some time.

[8] Names that stick out in my memory are Sebastian Rahtz and Thomas Esser, though I'm sure others have helped too.

continuing to bluntly disregard the LaTeX license details that stated that LaTeX had to be distributed as a whole.

It came to a climax with some messages making personal comments against the LaTeX developers, and in particular David. At this time I joined this debate by asking Richard if this person was really speaking for GNU, expressing my belief that if that was indeed the case, then this would show that the free software movement was in parts in a very sorry state. In a very personal email Richard replied with an apology for the damage that has been done in the name of the GNU project and in particular the verbal insults that some of the senior members of the LaTeX project team had to endure while defending some core rights for which the GNU project and the FSF were actually fighting. After some further discussions, to get to the bottom of the dispute he ensured that GNUTeX was taken off the archives and that was the end of it.

What I took away with me from that episode was that it is unfortunately very easy get carried away with the idea that "free" means that others have no rights worth preserving.

2.3 The issue with unmaintained packages

By the turn of the century many people thought that the fight over licenses in the TeX world was over. TeX and LaTeX and many other packages had licenses that were accepted as "free software licenses" by the FSF and more and more authors of packages with questionable licenses changed them or simply placed their work under the LPPL from the beginning.

However, we now started to realize that the LPPL in itself posed a problem for packages that lost their author/maintainer because he or she lost interest in the work or passed away as it sadly happened in some cases. As the LPPL required a name change in case of modification, a new maintainer of a previously unmaintained package couldn't fix bugs or extend the functionality without changing its name. In this way, perfectly good names could get lost for the LaTeX language — just because of the attempt to preserve the language integrity in the first place.

So in 2002 the idea was born to modify the LPPL once more by including a maintenance clause that would allow a person under certain circumstances (normally when a work was orphaned) to take over maintenance and in some sense ownership. The clause was written in a way such that it was up to the author of a work to allow or prevent this clause to apply.

On the whole I believe that this proposed license extension was a good idea as it further helped to stabilize LaTeX as a reliable language. But it had an unfortunate[9] side effect that everybody interested in free software licenses started to take another look at the LPPL.

The first person to ask was Richard, and I sent him a draft of the intended changes and explained their intentions. His reply was that he saw no issue with any of them.

2.4 The Debian dispute

While a new draft of the LPPL that contained a first version of the maintainers clause got discussed on the LaTeX project list one of its subscribers, Claire Connelly, explained that within the Debian community some people were unhappy about the LaTeX license and considered it nonfree (proposing to ban LaTeX from Debian distributions). She volunteered to initiate a license review on the Debian-legal list so that any issues with the license would be moved from the level of vague rumor to a level of facts that could be discussed intelligently.

However, a bit of a pity was that the first draft of LPPL-1.3 got presented which was not very clear, and thus added to the underlying misunderstandings rather than helping to clear them up. For example, one of the initial reviews remarked: "so my one-line summary of the license would be 'We hate forking'" which doesn't even remotely represent the intentions behind the license. Eventually, all such misunderstandings got resolved, but it took considerable effort. To be more precise, it took roughly 1600 messages (between July 2002 and June 2003) on the `debian-legal` list and elsewhere to sort them out and improve the wording to what in the end became LPPL-1.3.

In retrospect it turned out to be fairly difficult to identify the core reasons that led to this massive email exchange but from recollections there have been two major sources: textual and legal deficiencies in the license as such, and a head-on clash between different interpretations of "freedom". As a result, the discussions on Debian-legal were essentially with two only partly overlapping factions: a group of people who seriously wanted to understand the intentions behind the license and who where interested in providing guidance on how to improve it, while ensuring that it would meet the DFSG (Debian Free Software Guidelines), and a second group of people largely concerned about the rights of the programmer to modify code ad lib without any restrictions. The tenor here was "a requirement to rename is a

[9] Or perhaps fortunate when looking at the final outcome. However, if I would have known beforehand the amount of work that it took to get there, I would have let things lie.

restriction" and therefore unacceptable. In other words, an important requirement for "freedom" was the ability to modify some work in situ. For many situations this is an understandable requirement, e.g., when fixing bugs or when extending functionality. It does, however, become a bit blurry when modifications result in changing expected behavior. In that case one can argue that there is also the right of the recipient/user of the work to consider: to not be deliberately misled.

As an example, an interesting thread within the discussions spawned from a statement made by Boris Veytsman: "I am FOR the freedom of speech. However, I am against the freedom of my grocer to call a 950g weight 'a kilogram'." The replies were interesting and ranged from the opinion that the Debian project has no intention of supporting deliberate fraud (i.e., in this respect supporting his position) all the way to the claim that this would be acceptable behavior and needs to be supported in the spirit of freedom. Clearly there isn't any way to bridge the chasm between such widely different opinions with a license text and that was not attempted, though quite some energy was used on either side to argue the respective positions.

Leaving aside the discussions around the more extreme positions, the core point of dispute was the attempt of the LPPL to protect LaTeX as a language for interchange. By the nature of the beast this means acknowledging that in LaTeX file names of packages, classes, etc., are part of the (extensible) LaTeX language, i.e., each package extends or alters the language and its name together with its functionality becomes part of LaTeX when it is loaded. That is, by `\usepackage{`*name*`}` the user requests a certain package with a specific behavior upon which his document then relies.

To continue the above analogy, when a user loads the hypothetical package *weights* for typesetting grocery data then his document should not typeset 1kg at some installations, but 950g at others, as that would render LaTeX as a language useless.

In the end we settled for a toned down version of this requirement: although the LaTeX Project still strongly recommends a name change in case of modifications, the license alternatively allows for producing in situ modifications of components provided the derived work clearly and unambiguously identifies itself as a modified version of the original component to the user when used interactively in the same way the original component would identify itself to the user (Clause 6a of the license). In the case of a LaTeX package that would typically be achievable through an appropriate change of the `\ProvidesPackage` declaration. However, the LaTeX project team still recommends to use the name change approach and within the TeX community this is the predominantly used method. Whenever documents are intended to be exchanged this is the only safe way to ensure that your document typesets as intended and remains so over time. How powerful this approach is can be seen in the fact that with a few exceptions TeX and LaTeX documents from the last two decades can be still successfully typeset today.

Returning to the evolution of the license, on June 18th 2003 the Debian legal community agreed that LPPL 1.3 is a free software license with respect to the DFSG guidelines. This marked the endpoint of the active license development.

2.5 OSI certification

After the LPPL got accepted by Debian there was some discussion about submitting it also for approval through the Open Source Initiative, but effectively we had run out of steam. It would have meant (another) round of formal submissions and most likely discussions about wording and content and at least I didn't feel up to it at that time. But it was a somewhat naggingly open issue that the license wasn't certified by OSI, given that the LPPL codified the accepted behavior in a large and active free software community.

Fortunately, Will Robertson, a new member in the LaTeX project, took up that task and on the rather nice date 09/09/09 approval from the OSI was sought in the category: "Licenses that are popular and widely used or with strong communities".

As it turned out my fears of a repetition of the DFSG experience were groundless; it took about two dozen email exchanges to get the license accepted without any request for modification and only about two months later on Wednesday, November 11, 2009 the OSI Board formally approved it [8]. It then took nearly another year until the Open Source Initiative updated their web site, but in August 2010 the license finally appeared there [9].

3 Conclusions

From the history it seems fairly obvious that there are a good number of reasons why it is helpful to have a fairly uniform license landscape in a community like the TeX world. Does it have to be the LPPL? That question is less clear and as the discussions above have shown a matter of opinion and controversy. But on the whole I believe the answer is yes; the time and effort was well spent and the community has benefitted from it.

On the surface, languages like Perl or Python have issues similar as LaTeX. So why doesn't LaTeX use the GPL as they do? I guess the answer lies in the unique elements in the structure and usage of LaTeX (and perhaps its community?). It consists of a very large and complete collection of third-party packages in its standard form of distribution and all of this forms the language which the community expects to be able to use interchangeably. The other important difference is that for a library in Perl or Python that implements a set of functions it is normally well understood what represents a "correct" implementation of these functions, e.g., a time conversion function or a mathematical formula is either implemented correctly or not — but it is not going to be a matter of "taste".

In the area of typography, however, "correctness" has no reasonable definition and so it is not surprising that people have different opinion on what looks good or how something should be improved. This is perfectly acceptable and in fact encouraged throughout the community but it needs to be channeled in order to maintain the other important aspect of the language: support for interchange. And that is something that the GPL and similar licenses aren't able to ensure, while the LPPL approaches, and, as we think, resolved that issue adequately.

One of the commenters for the OSI review remarked, "I think this license is trying to create or enforce a policy for maintainership more than it concerns itself with copying & use/patent issues. I'm not convinced that this is a good idea, but TeX has an active community and if this license seems to work out for them, my level of concern isn't so great that I would second-guess that aspect." He is probably right, as in the TeX community the copying & use/patent issues play a minor role compared to resolving how to best maintain the language as a whole. Thus the idea of a maintainer and its rights is quite prominent. We definitely think it helps in that people know they are allowed to take over an orphaned work — it would probably happen less if it weren't stated explicitly as a possibility.

Is the LPPL the best solution to the issues that a community like the TeX community encounters? I guess not, but it is the best that we have been able to come up with after many (perhaps too many) hours and attempts.

3.1 Thanks

A number of people from the TeX community and from the Debian legal community have been instrumental in helping to make LPPL a better license and I would like to thank them all — in particular I would like to mention Jeff Licquia and Branden Robinson from Debian legal[10] with whom I had many fruitful discussions over ways to improve it and shape in a way that it properly expresses our intentions without conflicting with other people's thoughts on what comprises a free software license.

From the TeX community I'd like to thank all of my colleagues from the LaTeX project team, in particular David Carlisle and Chris Rowley who shouldered large proportions of the external discussions during those years. And a heartfelt thanks to Will Robertson who single-handedly got the license OSI-approved when the other team members had run out of steam to even attempt it.

Many other people from the TeX community contributed in one way or the other, be it on `latex-l`, `debian-legal`, or in private communication and it is impossible to list them all. As a representative of this huge group I should perhaps mention Boris Veytsman who wrote over one hundred messages on the subject during the debate with Debian.

Last not least I'd like to thank Richard Stallman for initiating the first round of discussions and drawing our intentions to the flaws of the initial license as well as opening at least my eyes to the complexity and difficulties behind free and open source software licensing.

◇ Frank Mittelbach
LaTeX Project
`http://www.latex-project.org`

A References

[1] The GNU GPL 1.0 can be found at: `www.gnu.org/licenses/old-licenses/gpl-1.0.txt`
[2] `en.wikipedia.org/wiki/Copyleft`
[3] License for TeX at the top of `mirror.ctan.org/systems/knuth/dist/tex/tex.web`
[4] Moral obligation clause for the `multicol` package at the top of `mirror.ctan.org/macros/latex/required/tools/multicol.dtx`
[5] The document "Modifying LaTeX": `mirror.ctan.org/macros/latex/doc/modguide.pdf`
[6] Comments on software licenses by the FSF: `www.gnu.org/licenses/license-list.html`
[7] `lists.debian.org/debian-legal/`
[8] `opensource.org/minutes2009111`
[9] `www.opensource.org/licenses/lppl`

[10] There are a few more people from Debian legal should perhaps be named but any list would naturally be incomplete. For those interested I suggest reading through the `debian-legal` archives [7] from that time; you will find this a lengthy but illuminating read in parts.

B The LaTeX Project Public License

LPPL Version 1.3c 2008-05-04

Preamble

The LaTeX Project Public License (LPPL) is the primary license under which the LaTeX kernel and the base LaTeX packages are distributed.

You may use this license for any work of which you hold the copyright and which you wish to distribute. This license may be particularly suitable if your work is TeX-related (such as a LaTeX package), but it is written in such a way that you can use it even if your work is unrelated to TeX.

The section 'WHETHER AND HOW TO DISTRIBUTE WORKS UNDER THIS LICENSE', below, gives instructions, examples, and recommendations for authors who are considering distributing their works under this license.

This license gives conditions under which a work may be distributed and modified, as well as conditions under which modified versions of that work may be distributed.

We, the LaTeX3 Project, believe that the conditions below give you the freedom to make and distribute modified versions of your work that conform with whatever technical specifications you wish while maintaining the availability, integrity, and reliability of that work. If you do not see how to achieve your goal while meeting these conditions, then read the document '`cfgguide.tex`' and '`modguide.tex`' in the base LaTeX distribution for suggestions.

Definitions

In this license document the following terms are used:

Work Any work being distributed under this License.

Derived Work Any work that under any applicable law is derived from the Work.

Modification Any procedure that produces a Derived Work under any applicable law – for example, the production of a file containing an original file associated with the Work or a significant portion of such a file, either verbatim or with modifications and/or translated into another language.

Modify To apply any procedure that produces a Derived Work under any applicable law.

Distribution Making copies of the Work available from one person to another, in whole or in part. Distribution includes (but is not limited to) making any electronic components of the Work accessible by file transfer protocols such as FTP or HTTP or by shared file systems such as Sun's Network File System (NFS).

Compiled Work A version of the Work that has been processed into a form where it is directly usable on a computer system. This processing may include using installation facilities provided by the Work, transformations of the Work, copying of components of the Work, or other activities. Note that modification of any installation facilities provided by the Work constitutes modification of the Work.

Current Maintainer A person or persons nominated as such within the Work. If there is no such explicit nomination then it is the 'Copyright Holder' under any applicable law.

Base Interpreter A program or process that is normally needed for running or interpreting a part or the whole of the Work.

A Base Interpreter may depend on external components but these are not considered part of the Base Interpreter provided that each external component clearly identifies itself whenever it is used interactively. Unless explicitly specified when applying the license to the Work, the only applicable Base Interpreter is a 'LaTeX-Format' or in the case of files belonging to the 'LaTeX-format' a program implementing the 'TeX language'.

Conditions on Distribution and Modification

1. Activities other than distribution and/or modification of the Work are not covered by this license; they are outside its scope. In particular, the act of running the Work is not restricted and no requirements are made concerning any offers of support for the Work.
2. You may distribute a complete, unmodified copy of the Work as you received it. Distribution of only part of the Work is considered modification of the Work, and no right to distribute such a Derived Work may be assumed under the terms of this clause.
3. You may distribute a Compiled Work that has been generated from a complete, unmodified copy of the Work as distributed under Clause 2 above, as long as that Compiled Work is distributed in such a way that the recipients may install the Compiled Work on their system exactly as it would have been installed if they generated a Compiled Work directly from the Work.
4. If you are the Current Maintainer of the Work, you may, without restriction, modify the Work, thus creating a Derived Work. You may also distribute the Derived Work without restriction, including Compiled Works generated from the Derived Work. Derived Works distributed in this manner by the Current Maintainer are considered to be updated versions of the Work.
5. If you are not the Current Maintainer of the Work, you may modify your copy of the Work, thus creating a Derived Work based on the Work, and compile this Derived Work, thus creating a Compiled Work based on the Derived Work.
6. If you are not the Current Maintainer of the Work, you may distribute a Derived Work provided the following conditions are met for every component

of the Work unless that component clearly states in the copyright notice that it is exempt from that condition. Only the Current Maintainer is allowed to add such statements of exemption to a component of the Work.

(a) If a component of this Derived Work can be a direct replacement for a component of the Work when that component is used with the Base Interpreter, then, wherever this component of the Work identifies itself to the user when used interactively with that Base Interpreter, the replacement component of this Derived Work clearly and unambiguously identifies itself as a modified version of this component to the user when used interactively with that Base Interpreter.

(b) Every component of the Derived Work contains prominent notices detailing the nature of the changes to that component, or a prominent reference to another file that is distributed as part of the Derived Work and that contains a complete and accurate log of the changes.

(c) No information in the Derived Work implies that any persons, including (but not limited to) the authors of the original version of the Work, provide any support, including (but not limited to) the reporting and handling of errors, to recipients of the Derived Work unless those persons have stated explicitly that they do provide such support for the Derived Work.

(d) You distribute at least one of the following with the Derived Work:

i. A complete, unmodified copy of the Work; if your distribution of a modified component is made by offering access to copy the modified component from a designated place, then offering equivalent access to copy the Work from the same or some similar place meets this condition, even though third parties are not compelled to copy the Work along with the modified component;

ii. Information that is sufficient to obtain a complete, unmodified copy of the Work.

7. If you are not the Current Maintainer of the Work, you may distribute a Compiled Work generated from a Derived Work, as long as the Derived Work is distributed to all recipients of the Compiled Work, and as long as the conditions of Clause 6, above, are met with regard to the Derived Work.
8. The conditions above are not intended to prohibit, and hence do not apply to, the modification, by any method, of any component so that it becomes identical to an updated version of that component of the Work as it is distributed by the Current Maintainer under Clause 4, above.
9. Distribution of the Work or any Derived Work in an alternative format, where the Work or that Derived Work (in whole or in part) is then produced by applying some process to that format, does not relax or nullify any sections of this license as they pertain to the results of applying that process.
10. (a) A Derived Work may be distributed under a different license provided that license itself honors the conditions listed in Clause 6 above, in regard to the Work, though it does not have to honor the rest of the conditions in this license.

    (b) If a Derived Work is distributed under a different license, that Derived Work must provide sufficient documentation as part of itself to allow each recipient of that Derived Work to honor the restrictions in Clause 6 above, concerning changes from the Work.
11. This license places no restrictions on works that are unrelated to the Work, nor does this license place any restrictions on aggregating such works with the Work by any means.
12. Nothing in this license is intended to, or may be used to, prevent complete compliance by all parties with all applicable laws.

No Warranty

There is no warranty for the Work. Except when otherwise stated in writing, the Copyright Holder provides the Work 'as is', without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Work is with you. Should the Work prove defective, you assume the cost of all necessary servicing, repair, or correction.

In no event unless required by applicable law or agreed to in writing will The Copyright Holder, or any author named in the components of the Work, or any other party who may distribute and/or modify the Work as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of any use of the Work or out of inability to use the Work (including, but not limited to, loss of data, data being rendered inaccurate, or losses sustained by anyone as a result of any failure of the Work to operate with any other programs), even if the Copyright Holder or said author or said other party has been advised of the possibility of such damages.

Maintenance of The Work

The Work has the status 'author-maintained' if the Copyright Holder explicitly and prominently states near the primary copyright notice in the Work that the Work can only be maintained by the Copyright Holder or simply that it is 'author-maintained'.

The Work has the status 'maintained' if there is a Current Maintainer who has indicated in the Work that they are willing to receive error reports for the Work (for example, by supplying a valid e-mail address). It is not

required for the Current Maintainer to acknowledge or act upon these error reports.

The Work changes from status 'maintained' to 'unmaintained' if there is no Current Maintainer, or the person stated to be Current Maintainer of the work cannot be reached through the indicated means of communication for a period of six months, and there are no other significant signs of active maintenance.

You can become the Current Maintainer of the Work by agreement with any existing Current Maintainer to take over this role. If the Work is unmaintained, you can become the Current Maintainer of the Work through the following steps:

1. Make a reasonable attempt to trace the Current Maintainer (and the Copyright Holder, if the two differ) through the means of an Internet or similar search.
2. If this search is successful, then enquire whether the Work is still maintained.
   (a) If it is being maintained, then ask the Current Maintainer to update their communication data within one month.
   (b) If the search is unsuccessful or no action to resume active maintenance is taken by the Current Maintainer, then announce within the pertinent community your intention to take over maintenance. (If the Work is a LaTeX work, this could be done, for example, by posting to `comp.text.tex`.)
3. (a) If the Current Maintainer is reachable and agrees to pass maintenance of the Work to you, then this takes effect immediately upon announcement.
   (b) If the Current Maintainer is not reachable and the Copyright Holder agrees that maintenance of the Work be passed to you, then this takes effect immediately upon announcement.
4. If you make an 'intention announcement' as described in 2b above and after three months your intention is challenged neither by the Current Maintainer nor by the Copyright Holder nor by other people, then you may arrange for the Work to be changed so as to name you as the (new) Current Maintainer.
5. If the previously unreachable Current Maintainer becomes reachable once more within three months of a change completed under the terms of 3b or 4, then that Current Maintainer must become or remain the Current Maintainer upon request provided they then update their communication data within one month.

A change in the Current Maintainer does not, of itself, alter the fact that the Work is distributed under the LPPL license.

If you become the Current Maintainer of the Work, you should immediately provide, within the Work, a prominent and unambiguous statement of your status as Current Maintainer. You should also announce your new status to the same pertinent community as in 2b above.

Whether and How to Distribute Works under This License

This section contains important instructions, examples, and recommendations for authors who are considering distributing their works under this license. These authors are addressed as 'you' in this section.

Choosing This License or Another License

If for any part of your work you want or need to use *distribution* conditions that differ significantly from those in this license, then do not refer to this license anywhere in your work but, instead, distribute your work under a different license. You may use the text of this license as a model for your own license, but your license should not refer to the LPPL or otherwise give the impression that your work is distributed under the LPPL.

The document '`modguide.tex`' in the base LaTeX distribution explains the motivation behind the conditions of this license. It explains, for example, why distributing LaTeX under the GNU General Public License (GPL) was considered inappropriate. Even if your work is unrelated to LaTeX, the discussion in '`modguide.tex`' may still be relevant, and authors intending to distribute their works under any license are encouraged to read it.

A Recommendation on Modification Without Distribution

It is wise never to modify a component of the Work, even for your own personal use, without also meeting the above conditions for distributing the modified component. While you might intend that such modifications will never be distributed, often this will happen by accident – you may forget that you have modified that component; or it may not occur to you when allowing others to access the modified version that you are thus distributing it and violating the conditions of this license in ways that could have legal implications and, worse, cause problems for the community. It is therefore usually in your best interest to keep your copy of the Work identical with the public one. Many works provide ways to control the behavior of that work without altering any of its licensed components.

How to Use This License

To use this license, place in each of the components of your work both an explicit copyright notice including your name and the year the work was authored and/or last substantially modified. Include also a statement that the distribution and/or modification of that component is constrained by the conditions in this license.

Here is an example of such a notice and statement:

```
%% pig.dtx
%% Copyright 2005 M. Y. Name
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3 or later is part of all distributions of LaTeX
% version 2005/12/01 or later.
%
% This work has the LPPL maintenance status 'maintained'.
%
% The Current Maintainer of this work is M. Y. Name.
%
% This work consists of the files pig.dtx and pig.ins
% and the derived file pig.sty.
```

Given such a notice and statement in a file, the conditions given in this license document would apply, with the 'Work' referring to the three files '`pig.dtx`', '`pig.ins`', and '`pig.sty`' (the last being generated from '`pig.dtx`' using '`pig.ins`'), the 'Base Interpreter' referring to any 'LaTeX-Format', and both 'Copyright Holder' and 'Current Maintainer' referring to the person 'M. Y. Name'.

If you do not want the Maintenance section of LPPL to apply to your Work, change 'maintained' above into 'author-maintained'. However, we recommend that you use 'maintained' as the Maintenance section was added in order to ensure that your Work remains useful to the community even when you can no longer maintain and support it yourself.

Derived Works That Are Not Replacements

Several clauses of the LPPL specify means to provide reliability and stability for the user community. They therefore concern themselves with the case that a Derived Work is intended to be used as a (compatible or incompatible) replacement of the original Work. If this is not the case (e.g., if a few lines of code are reused for a completely different task), then clauses 6b and 6d shall not apply.

Important Recommendations

Defining What Constitutes the Work

The LPPL requires that distributions of the Work contain all the files of the Work. It is therefore important that you provide a way for the licensee to determine which files constitute the Work. This could, for example, be achieved by explicitly listing all the files of the Work near the copyright notice of each file or by using a line such as:

```
% This work consists of all files listed in manifest.txt.
```

in that place. In the absence of an unequivocal list it might be impossible for the licensee to determine what is considered by you to comprise the Work and, in such a case, the licensee would be entitled to make reasonable conjectures as to which files comprise the Work.

siunitx: A comprehensive (SI) units package

Joseph Wright

Abstract

The siunitx package provides a powerful toolkit for typesetting numbers and units in LaTeX. By incorporating detail about the agreed rules for presenting scientific data, siunitx enables authors to concentrate on the meaning of their input and leave the package to deal with the formatting. Version 2 of siunitx increases the scope of the package and is intended to act as a solid base for further development.

1 Introduction

Physical quantities are important mathematical data, which appear in print in many scientific publications. These physical quantities are made up of a number and an associated unit: the two parts together make up a single mathematical entity. A series of international agreements have led to the *Système International d'Unités* (SI units), administered by the *Bureau International des Poids et Mesures* (Bureau International des Poids et Mesures, 2010). This system lays down units such as the kilogram, metre and kelvin, and provides a self-consistent approach to measuring all physical quantities. At the same time, there are clearly defined standards which describe how the data should be presented. The US National Institute for Standards and Technology (NIST) have described 'best practices' for presenting numbers and units in scientific contexts (National Institute for Standards and Technology, 2010).

LaTeX's approach of providing logical structure is ideally suited to helping authors follow these rules without needing to study them in detail. However, this does not mean that it has been easy to write a comprehensive package to deal with both numbers and units. This article provides an overview of the siunitx package (Wright, 2010), which aims to be *A comprehensive (SI) units package*.

2 A little history

2.1 Before siunitx

siunitx is the latest in a series of different LaTeX packages for handling units, and it is therefore useful to know a little about the earlier implementations.

The package units (Reichert, 1998) provides perhaps the most basic units support: the macro `\unit` to mark up input as a unit (with an optional value).

`\unit[`⟨*value*⟩`]{`⟨*unit*⟩`}`

Building on this, the unitsdef package (Happel, 2005) provides a number of pre-defined unit macros, which expand to the appropriate symbol(s) for the unit. Unfortunately, these definitions require additional macros so that the package can format them correctly:

```
\newunit{\newtonmeter}
  {\newton\unittimes\meter}
\newunit{\newtonmeterpersec}
  {\per{\newton\unittimes\meter}{\second}}
```

As we will see, siunitx is able to define similar unit macros but without needing support functions such as `\unittimes`.

An alternative approach to defining unit macros was provided by SIunits (Heldoorn and Wright, 2007). SIunits provides a larger set of pre-defined units than unitsdef, but again requires support functions in these definitions. These support functions define the appearance of units, so altering how a unit is displayed requires a new definition. For example, `\amperepersquaremetre` prints $\mathrm{A/m^2}$ while `\amperepersquaremetrenp` is used for $\mathrm{A\,m^{-2}}$.

The SIstyle package (Els, 2008) tackles the need for units and values to be typeset using the correct fonts. As such, it focusses on the appearance of the output, rather than logical markup of input. This can have advantages, as input such as

```
\SI{10}{m/s}
```

is certainly easy to read.

Finally, while not focussed on units, the numprint package (Harders, 2008) has provided the complementary ability to format numbers, for example separating large numbers so that the digits are grouped.

2.2 A new approach to units

Before the development of siunitx, the best approach to typesetting units was to use the combination of SIunits and SIstyle, to get logical input and controlled output.

Development of siunitx began with a simple bug report for SIunits on `comp.text.tex`. I naïvely volunteered to take a look at it, and contacted Marcel Heldoorn with a solution to the issue at hand. It turned out that he no longer had time for supporting SIunits, and said that I was free to take over. Having fixed the bug at hand, I even more naïvely asked on the newsgroup if there were any improvements to be made. I soon had quite a list!

I took a step back, and looked at the combination of SIunits and SIstyle and the feature request list I'd built up. It was clear that I needed to do more than simply patch SIunits. I also took a careful look at biblatex (Lehman, 2010), which shows how a user interface should be done. My conclusion was that I needed to write a package from the ground up, combining the features of SIunits and SIstyle with

a key–value interface rather than a complex mix of different control macros.

This effort led to the first version of siunitx, which inherited a great deal of code from its predecessors. The feature requests kept coming, and some of these were rather 'bolted on' to the first version of siunitx. Partly as a result of this, and partly as I'm now involved in the LaTeX3 Project, I decided to rewrite the package using the expl3 approach (LaTeX3 Project, 2010). This has allowed the internal code of the package to be made much more structured, which will hopefully enable me to continue to add new features without compromising the existing features of the package.

3 Units

The core function of the siunitx package is typesetting units in a flexible way and with a natural input syntax. The macro which does this is `\si` (think 'a bit like "SI"'). The `\si` macro can typeset both literal input such as `\si{m.s^{-1}}` and the semantic version `\si{\metre\per\second}` to give the same output: $\mathrm{m\,s^{-1}}$. Allowing two forms of input means that users get to make a choice on how semantic they want their input to be.

There are lots of things going on when something like `\si{m.s^{-1}}` is typeset. The first thing to notice is that the superscript will work equally-happily in math and text mode (the same is true for subscripts). What is also true is that you get *exactly the same* output in both cases: the fonts and spacing used are determined by siunitx. The standard settings use the document text font for units, but the document math font for numbers. Numbers as handled by siunitx are essentially mathematical, and so they should (probably) match any other mathematics. Both numbers and units are typeset ignoring any local font changes, such as bold or italics.

Now, some of those choices will not suit everyone: a classic case is units in section headings, where bold seems a more 'natural' choice than the usual mid-weight font. To handle the need to be flexible, siunitx provides the `\sisetup` macro, which takes a list of key–value options (there are currently about 140!). Settings can also be given as an optional argument to `\si`, which allows them to be applied to individual cases: `\sisetup` applies to everything that follows, subject to the usual TeX grouping. So in a heading, rather than `\si{m.s^{-1}}` we might have `\si[detect-weight]{m.s^{-1}}`.

What about the unit macros: are they flexible? One of the key aims of siunitx is to use semantic markup with units so that different output appearances don't need different input syntaxes. Sticking with the example `\si{\metre\per\second}`, there are a number of possible variations. As we've already seen, the standard settings give '$\mathrm{m\,s^{-1}}$', with the `\per` macro converted into a superscripted power. Another common choice is 'm/s', using a slash to show the division. That's done by setting the option `per-mode = symbol`. Then again, you might want to show things as a fraction, '$\frac{\mathrm{m}}{\mathrm{s}}$', achieved by setting `per-mode = fraction`.

That is fine for a relatively simple unit, but what about a more complex case such as

```
\si{\joule\per\mole\squared
  \metre\per\cubic\candela}
```

(*i.e.* $\mathrm{J\,mol^{-2}\,m\,cd^{-3}}$)? When given as a fraction or using a slash, there need to be some brackets or rearrangement of the order. The package knows about this, and can automatically produce the appropriate output, which might be '$\mathrm{J\,m/(mol^2\,cd^3)}$' or '$\frac{\mathrm{J\,m}}{\mathrm{mol^2\,cd^3}}$'. It can even produce mathematically-invalid output like '$\mathrm{J/mol^2\,m/cd^3}$' if you want.

As already indicated, there are a *lot* of options available, and I don't want to repeat the manual here. However, I hope that the concept of 'one clear input, many forms of output' comes through.

One last idea to highlight is that new units can be defined using the two macros `\DeclareSIUnit` and `\DeclareSIUnitWithOptions`. These are used to set up new units, perhaps with a special appearance. So if I want to give '$\frac{\mathrm{m}}{\mathrm{s}}$' with a slash but everything else as powers, I might define

```
\DeclareSIUnitWithOptions{\mpers}
  {\metre\per\second}{per-mode = fraction}
```

and then use `\mpers` as the unit. Name clashes are not an issue: siunitx only defines the unit macros within the argument of its own macros.

4 Numbers

Most of the time, units in scientific writing go with numbers. So siunitx needs to be able to deal with those as well. This is handled by the imaginatively-named `\num` macro. This takes the number itself as the one mandatory argument, with a first optional argument for any settings that apply.

Probably the most common function this performs is grouping digits. So `\num{12345}` will give '12 345' rather than '12345'. The latter is of course available as an option: `group-digits = false`.

There are two other common formatting changes. First, it is useful to format `\num{12e3}` as '12×10^3', which is done automatically. Secondly, depending on where in the world you are you might want `\num{123.45}` to display as '123,45'. The package uses settings such as `input-exponent-markers` and

`output-decimal-marker` to decide on the format of the input and how it should look as output for these cases.

Another common requirement with numbers is to round them, fixing either decimal places or significant figures. Here, the two options `round-mode` and `round-precision` are used. The standard settings do not do any rounding at all, so `\num{123.456}` gives '123.456'. This can easily be converted into '123.46' by setting `round-mode = places`, or '120' by setting `round-mode = figures`. As you might work out, the standard setting is `round-precision = 2`, and this applies to whatever rounding is being done. As we'll see, rounding is particularly useful in tables.

5 Numbers with units

We've seen both numbers and units on their own, but obviously the two need to be combined. For that, the `\SI` macro is available, and takes one number and one mandatory unit argument to print the combination of the two. As with `\num` and `\si`, the first argument is optional and applies local settings.

All of the options for units and numbers alone apply to combinations too, but there are some special options which only make sense for combinations. The most obvious is a choice of the separator between a number and the associated unit. The standard setting is thin space: '10 m'. This is controlled by the `number-unit-separator` option, which expects an argument in math mode. So to use a full text-mode space you'd set `number-unit-separator = \text{ }`, with the result '10 m'.

Closely related to the *size* of the space between number and unit is how it behaves at a line break. The standard settings do not allow a break here, but particularly in narrow columns (such as in this document) it is more realistic to allow breaks to occur. The option to do control is called `allow-number-unit-breaks`, which will allow a break: '10 m'. (As you might guess, the text in this paragraph is finely balanced to give a break in the right place!).

6 Tables

Placing numbers in tables so that the decimal markers are aligned is very important so that scientific data are clear. To support this, siunitx defined the `S` column type. At its most basic, all you do is use this in place of a `c` column and let the package do the work. So with the input

```
\begin{tabular}{S}
  \toprule
  {Some numbers} \\
  \midrule
    1.234e2 \\
  567.8e0   \\
    4.3543e1 \\
  \bottomrule
\end{tabular}
```

you can get the output in Table 1. Notice that the table heading is wrapped in braces: this tells siunitx to treat this as text and not to look for a number.

Table 1: Simple number alignment using the `S` column

Some numbers
1.234×10^2
567.8
4.3543×10^1

Now, Table 1 is not a very good table, as the data are not really comparable. It's usually best to avoid exponents in the body of a table, and to put them into the header instead. It's also common to round tabular data to some sensible number of significant figures. Table 2 is a better version of the same table, with input that reads

Table 2: Exploiting the power of the `S` column

Some numbers/10^2
1.23
5.68
0.44

```
\begin{tabular}{S[
    table-auto-round,
    table-omit-exponent,
    table-format   = 1.2,
    fixed-exponent = 2
  ]}
  \toprule
  {Some numbers/\num{e2}} \\
  \midrule
    1.234e2 \\
  567.8e0   \\
    4.3543e1 \\
  \bottomrule
\end{tabular}
```

This illustrates several table-related functions in one go. First, the `S` column accepts an optional argument, letting each column have its own behaviour. The option `table-format` is used to define how much space siunitx will need for the output: here there is one integer and two decimal digits, with no signs or exponents. The `table-auto-round` and `table-omit-exponent` options have self-explanatory

names, while `fixed-exponent = 2` is used to 'shift' the decimal place in the input. This combination of options means that the input does not require any manipulation: an advantage if it's a long list copied from some automated source!

7 Extras

We've seen the main macros that siunitx provides, but there are a few more specialist ones which deal with more unusual cases. These 'extras' all take the usual optional first argument, and have their own dedicated options.

The `\ang` macro takes angle input, either as a decimal or in degrees, minutes and seconds. The latter is necessary for things like '1°2′3″', which is given as `\ang{1;2;3}`. One particularly notable option here is `angle-symbol-over-decimal`, which can give output such as '1°2′3.″4' from the input

```
\ang[angle-symbol-over-decimal]{1;2;3.4}
```

I'm told that this is useful for astronomy: that is far from my area of expertise, but as always the aim is to give users what they want with the minimum of fuss.

There are two specialist macros for cases where the same unit applies to multiple numbers: `\SIrange` and `\SIlist`. These let you type

```
\SIrange{10}{20}{\metre}
```

and get '10 m to 20 m', or to type

```
\SIlist{1;2;3;4;5}{\metre}
```

and get '1 m, 2 m, 3 m, 4 m and 5 m'. You'll notice that the standard settings repeat the unit for each number. Not everyone will like that, so you can use

```
\SIlist[list-units = brackets]
  {1;2;3;4;5}{\metre}
```

and get '(1, 2, 3, 4 and 5) m', or even

```
\SIlist[list-units = single]
  {1;2;3;4;5}{\metre}
```

to give the (mathematically incorrect) '1, 2, 3, 4 and 5 m'.

8 Summary

The siunitx package aims to be 'a comprehensive (SI) units package' while remaining accessible to users. It supplies a small number of flexible macros along with a large set of key–value options to control output either globally or for individual cases.

Here, I've tried to highlight how siunitx works, showing off some of the powerful features it supplies. The manual contains examples for almost all of the options, and if you can't see how to do something with siunitx you can always submit a feature request!

9 Acknowledgements

Thanks to Danie Els and Marcel Heldoorn for the SIstyle and SIunits packages: siunitx would not exist without them. Thanks to Stefan Pinnow for his careful testing of siunitx: his contribution to the package has been invaluable.

References

LaTeX3 Project. "The expl3 package and LaTeX3 programming". Available from CTAN, `macros/latex/contrib/expl3`, 2010.

Bureau International des Poids et Mesures. "The International System of Units (SI)". `http://www.bipm.org/en/si/`, 2010.

Els, D. N. J. "The SIstyle package". Available from CTAN, `macros/latex/contrib/SIstyle`, 2008.

Happel, Patrick. "unitsdef – Typesetting units with LaTeX 2ε". Available from CTAN, `macros/latex/contrib/unitsdef`, 2005.

Harders, Harald. "The numprint package". Available from CTAN, `macros/latex/contrib/numprint`, 2008.

Heldoorn, Marcel, and J. A. Wright. "The SIunits package: Consistent application of SI units". Available from CTAN, `macros/latex/contrib/SIunits`, 2007.

Lehman, Philipp. "The biblatex package: Programmable Bibliographies and Citations". Available from CTAN, `macros/latex/contrib/biblatex`, 2010.

National Institute for Standards and Technology. "International System of Units from NIST". `http://physics.nist.gov/cuu/Units/index.html`, 2010.

Reichert, Axel. "units.sty – nicefrac.sty". Available from CTAN, `macros/latex/contrib/units`, 1998.

Wright, Joseph A. "siunitx – A comprehensive (SI) units package". Available from CTAN, `macros/latex/contrib/siunitx`, 2010.

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

Glisterings

Peter Wilson

> Fame is no plant that grows on mortal soil,
> Nor in the glistering foil
> Set off to the world, nor in broad rumour lies:
> But lives and spreads aloft by those pure eyes
> And perfect witness of all-judging Jove;
> As he pronounces lastly on each deed,
> Of so much fame in heaven expect thy meed.
>
> *Lycidas, Elegy on a Friend drowned in the Irish Channel, 1637*, John Milton

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.

Corrections, suggestions, and contributions will always be welcome.

> I wuz framed!
>
> *Traditional criminal defence*

1 Framing

People ask on `comp.text.tex` about putting boxes around some text, or putting a background colour behind some text. An `\fbox` puts a frame round its contents and a `\colorbox`, from the (x)color package, makes a coloured background. However, these boxes don't continue across page breaks and on occasion the text for framing will have a page break in the middle. Donald Arseneau's framed package [1] is often the solution for this. The package provides four environments as demonstrated below.

The `framed` environment from the framed package puts a frame around its contents. Unlike the usual box commands or environments, framed continues over page breaks.

The `shaded` environment, also from the framed package, puts a colour (or a shade of gray) behind its contents which may continue across page breaks. Before using this environment you have to define a colour named `shadecolor`. In this case I have used the xcolor package and

```
\definecolor{shadecolor}{gray}{0.8}
```

FRAMED ILLUSTRATION

Figure 1: Example framed figure

FRAMED ILLUSTRATION & CAPTION

Figure 2: Example framed figure and caption

The `snugshade` environment is similar to `shaded` and puts a colour (or a shade of gray) behind its contents which may continue across page breaks. Unlike the `shaded` environment, though, the colour does not bleed into the margins. This time I set `shadecolor` as

```
\definecolor{shadecolor}{gray}{0.9}
```

to give a look not so arresting as with the setting for the `shaded` example.

The `leftbar` environment, also from the framed package, puts a vertical line at the left side of its contents, which may continue across page breaks.

The package can be useful even if you know that there won't be a page break. For instance a `figure` is constrained to be on one page, but you may want to set off some illustrations from the text.

Figure 1 is produced from the code below; only the illustration is framed, not the caption.

```
\begin{figure}
\centering
\begin{framed}\centering
FRAMED ILLUSTRATION
\end{framed}
\vspace*{-0.5\baselineskip}
\caption{Example framed figure}
\label{fig:framef}
\end{figure}
```

Figure 2, where the entire `figure` environment including the caption is framed, is produced from the code below:

```
\begin{figure}
\begin{framed}\centering
FRAMED ILLUSTRATION \& CAPTION
\caption{Example framed figure and caption}
\label{fig:framefcap}
\end{framed}
\end{figure}
```

If a full frame seems a bit of an overkill then using rules might be appropriate. Figure 3 is produced from the following code, and no package is needed. For illustrative purposes I have used very thick rules

RULED ILLUSTRATION & CAPTION

Figure 3: Ruled figure and caption (2)

SHADED ILLUSTRATION & CAPTION

Figure 4: Example shaded figure and caption

above and below the illustration; normally I would not have them so bold.

```
\begin{figure}
\centering
\rule{\linewidth}{2pt}\\
\vspace*{0.5\baselineskip}
RULED ILLUSTRATION \& CAPTION
\vspace{0.5\baselineskip}
\hrule
\caption{Ruled figure and caption (2)}
\label{fig:ruledfcap}
\rule{\linewidth}{2pt}
\end{figure}
```

Depending on its contents, using shading to delineate an illustration may be an option. Figure 4 is produced by the following code, which uses the framed package's `shaded` environment.

```
\begin{figure}
\begin{shaded}\centering
SHADED ILLUSTRATION \& CAPTION
\caption{Example shaded figure and caption}
\label{fig:shadefcap}
\end{shaded}
\end{figure}
```

The framed package does have some limitations on what can be included within its environments. floats, footnotes, marginpars and header entries will be lost. Further, the package does not work with the multicol package's page breaking, or other code to perform multicolumn balancing. Some restrictions may be lifted in later versions of the package.

Using the package you can create your own new cross-page environments, based on the `MakeFramed` environment defined in the package and specifying a `\FrameCommand` macro which should draw the frame. The `MakeFramed` environment takes one argument which should contain any adjustments to the text width, applied to `\hsize`, and some form of a *restore* command, such as the package's `\FrameRestore` macro or the LaTeX internal `\@parboxrestore`, that restores the text attributes to their normal state. The length `\width` is the width of the frame itself. Some examples are given later. But first...

The frame in the `framed` environment is implemented as an `\fbox` via:

```
\providecommand{\FrameCommand}{%
  \setlength{\fboxrule}{\FrameRule}%
  \setlength{\fboxsep}{\FrameSep}%
  \fbox}
```

where `\FrameRule` and `\FrameSep` are lengths defined by the package. By changing these you can change the rule thickness and spacing of the frame. Here is the definition of the `framed` environment itself, which uses the default `\FrameCommand` defined above.

```
\newenvironment{framed}{%
  {\MakeFramed {\advance\hsize-\width
   \FrameRestore}}%
  {\endMakeFramed}
```

where `\FrameRestore` restores some text settings, but not as many as have to be done at the end of a `minipage`.

The other environments are defined similarly. Both `shaded` and `snugshade` use a `\colorbox` as the framing mechanism.

```
\newenvironment{shaded}{%
  \def\FrameCommand{\fboxsep=\FrameSep
  \colorbox{shadecolor}}%
  \MakeFramed {\FrameRestore}}%
  {\endMakeFramed}
\newenvironment{snugshade}{%
  \def\FrameCommand{%
  \colorbox{shadecolor}}%
  \MakeFramed {\FrameRestore\@setminipage}}%
  {\par\unskip\endMakeFramed}
```

The `leftbar` environment simply uses a vertical rule.

```
\newenvironment{leftbar}{%
  \def\FrameCommand{\vrule width 3pt
                    \hspace{10pt}}%
  {\MakeFramed {\advance\hsize-\width
   \FrameRestore}}%
  {\endMakeFramed}
```

Note that in the `framed` and `leftbar` environments the text is narrower than the normal measure, while in the shade environments the text width is unaltered and the shading extends into the margins.

> Tyger, tyger, burning bright
> In the forests of the night,
> What immortal hand or eye
> Could frame thy fearful symmetry?
>
> *Songs of Experience*, William Blake

2 New frames

In some cases it is relatively easy to define your own framing environment based on the framed package,

but I have found that some experimentation is often required.

Perhaps you would like to center and frame some text. Here's how.

```
\newenvironment{narrowframe}[1][0.8\hsize]%
  {\MakeFramed{\setlength{\hsize}{#1}
   \FrameRestore}}%
  {\endMakeFramed}
```

This is the narrowframe environment where you can adjust the width with the environment's optional length argument. This example is set with the default width.

Or perhaps you would like something a little smoother:

This is the roundedframe environment. It requires the fancybox package. Perhaps you could use something other than the \ovalbox box from that package to give a different frame.

This is the definition I have used for the environment. You can, of course, change the lengths to suit.

```
\newenvironment{roundedframe}{%
  \def\FrameCommand{%
    \cornersize*{20pt}%
    \setlength{\fboxsep}{5pt}%
    \ovalbox}%
  \MakeFramed{\advance\hsize-\width
              \FrameRestore}}
  {\endMakeFramed}
```

Another request that pops up from time to time on comp.text.tex is for an environment to show off examples or questions. Robert Nyqvist [2] answered one of these by providing code based on the framed package, that was basically as follows. An example of the ruledexample environment is shown below as Ruled Example 1.

```
\makeatletter
\definecolor{rulecolor}{gray}{0.65}
\newcounter{ruledexample}
\newlength{\releftgap}
  \setlength{\releftgap}{4pt}
\newlength{\rerightgap}
  \setlength{\rerightgap}{1em}
\newlength{\rerule}
  \setlength{\rerule}{1.25pt}
\newlength{\Eheight}
\newenvironment{ruledexample}[1][Example]{%
  \settoheight{\Eheight}{\textbf{#1}}%
  \addtolength{\Eheight}{-\rerule}%
  \def\FrameCommand{\hspace{-\releftgap}%
    {\color{rulecolor}%
     \vrule width \rerule}%
    \hspace{\releftgap}\hspace{-\rerule}}%
  \MakeFramed{\advance\hsize-\width}%
  \refstepcounter{ruledexample}%
  \makebox[0pt][l]{%
    \hspace{-\parindent}%
    \hspace{\rerightgap}%
    {\color{rulecolor}\rule{1.5em}{\rerule}}%
    \quad
    \raisebox{-0.5\Eheight}[0pt]{%
      \textbf{#1\ \theruledexample}}%
   }\\[.5\baselineskip]%
    \noindent\ignorespaces}%
  {\@afterheading\\
   \makebox[0pt][l]{%
     \hspace{-\releftgap}%
     {{\color{rulecolor}
       \rule{\columnwidth}{\rerule}%
       \rule{\releftgap}{\rerule}}%
   }}
  \endMakeFramed}
\makeatother
```

Ruled Example 1

This is the ruledexample environment, which is titled and numbered and can be \labelled, and which will break across pages if need be. The code basis was originally posted to comp.text.tex by Robert Nyqvist in 2003 but I have added some extra means of controlling the rules and spacing.

- rulecolor is the color of the rules
- \rerule is the thickness of the rules
- \releftgap is the distance the vertical rule is moved into the margin
- \rerightgap is the indentation of the title rule
- You can use the optional argument to specify the title, which by default is 'Example'.

As you can see by this example, you can use code like itemize within a framed environment.

Some users have found that the behaviour of the framed environment within a list environment such as quote, or adjustwidth from the changepage package [4], is not quite what they expected. This is described in a little more detail below

> This is the start of a quotation environment. You can use the framed environment within this, and also any other list based environment.

We are now in the framed environment, and notice how the frame extends across the original textwidth. This is not always what you might want.

Donald Arseneau and I had an exchange of views, and code, on this and we each produced code that created a closer frame. As usual Donald's code was much better than mine.

> Now we are in the qframe environment based on Donald Arseneau's code and which I included in my memoir class [3]. The class also provides a 'shaded' version.

If you happen to use the adjustwidth environment, from the changepage package[1] [4] or the memoir class, then the qframe environment works better in it than does the framed environment.

Here endeth the quotation environment.

This is the definition of the qframe environment:

```
\makeatletter
\newenvironment{qframe}{%
  \def\FrameCommand##1{%
    \setlength{\fboxrule}{\FrameRule}%
    \setlength{\fboxsep}{\FrameSep}%
    \hskip\@totalleftmargin\fbox{##1}%
    \hskip-\linewidth
    \hskip-\@totalleftmargin
    \hskip\columnwidth}%
  \MakeFramed{\advance\hsize-\width
              \advance\hsize \FrameSep
              \@totalleftmargin\z@
              \linewidth=\hsize}}%
  {\endMakeFramed}
\makeatother
```

If you put this code in a class (.cls) or package (.sty) file then you do not need the \makeatletter and \makeatother pairing, otherwise you do.

Another potentially useful sort of frame is where there is a title — one that gets repeated after a page break. The code for this is considerably more complicated than earlier, and I had to do quite a lot of experimentation to get it to work. First, though, an example.

A 'framewithtitle'

Using the framewithtitle environment, which this is, you can put a title inside the frame. If the contents extend past a pagebreak then a 'continued' title is put at the start of the frame on the succeeding pages. The code for framewithtitle is given in this article.

The position of the title is set by the \titleframe command which is initially defined as below. If you would prefer the title to be centered, then change it like:

```
\newcommand*{\frametitle}[1]{%
  \strut#1}%                    at left
\renewcommand*{\frametitle}[1]{%
  \centerline{\strut#1}}%       centered
```

This example shows that you can include verbatim text in a framed environment.

Here is the code for the framewithtitle environment. It is based on hints from Donald Arseneau that applied to an earlier version of framed. A lot of the code is concerned with handling the title so that the first can be used initially, then after a page break a continuation title will be used instead. I have also extracted a lot of code from my original version as separate macros as I'm going to reuse much of it later. My most difficult problem was that internally the package keeps resetting the text to determine where any page breaking will occur, and only after that does it actually typeset; I couldn't assume that the initial title would be immediately set and had to reserve its typesetting until after the all the internal resetting had been completed; I used \ifcontframe for this purpose. The macro \Fr@meSetup does most of the work related to the titling while \FrameTitle actually typesets the title(s). The required argument to the framewithtitle environment is the initial title, and you can use the optional argument to override the default continuation title.

```
\makeatletter
\newcommand*{\frametitle}[1]{\strut#1}
\newif\ifcontframe

\newcommand*{\Fr@meSetup}[2]{%
  \fboxrule=\FrameRule \fboxsep=\FrameSep
  \global\contframefalse
  \def\Fr@meFirst{\textbf{#2}}%
  \def\Fr@meCont{\textbf{#1}}%
  \def\FrameCommand##1{%
    \Title@Fr@me{\Fr@meCurrent}{##1}%
    \global\let\Fr@meCurrent\Fr@meNext
    \ifcontframe
      \global\let\Fr@meNext\Fr@meCont
    \fi
    \global\contframetrue}%
  \global\let\Fr@meCurrent\Fr@meFirst
  \global\let\Fr@meNext\Fr@meFirst}

\newcommand*{\FrameTitle}[1]{%
  \nobreak \vskip -0.7\FrameSep
    \rlap{\frametitle{#1}}%
    \nobreak\nointerlineskip
    \vskip 0.7\FrameSep}
```

[1] changepage is the successor to the chngpage package which is being phased out.

```
\newenvironment{framewithtitle}[2]%
               [\Fr@meFirst\ (cont.)]{%
  \def\Title@Fr@me##1##2{%
    \fbox{\vbox{\FrameTitle{##1}%
    \hbox{##2}}}}%
  \Fr@meSetup{#1}{#2}%
  \MakeFramed{%
    \advance\hsize-\width
    \FrameRestore}}%
  {\global\contframefalse
   \endMakeFramed}
\makeatother
```

As an alternative to `framewithtitle` you can use the `titleframed` environment where the title is set outside the frame. The arguments to the two environments are the same, as is setting the title's horizontal position.

A 'titledframe'

With the `titledframe` environment, which this is, you can put a title on top of the frame. If the contents extend past a pagebreak then a 'continued' title is put at the start of the frame on the succeeding pages. The code for `titledframe` is given in this article.

The position of the title is set by the `\titleframe` command which is initially defined as below. If you would prefer the title to be centered, then change it like:

```
\newcommand*{\frametitle}[1]{%
  \strut#1}%              at left
\renewcommand*{\frametitle}[1]{%
  \centerline{\strut#1}}% centered
```

which is what I have done for this example.

So, here is the code for `titledframe` which, as you can see, shares most of the code with its companion environment.

```
\makeatletter
\newenvironment{titledframe}[2]%
               [\Fr@meFirst\ (cont.)]{%
  \def\Title@Fr@me##1##2{%
    \vbox{\FrameTitle{##1}%
      \noindent\fbox{##2}}}
  \Fr@meSetup{#1}{#2}%
  \MakeFramed{%
    \advance\hsize-\width
    \advance\hsize -2\FrameRule
    \advance\hsize -2\FrameSep
    \FrameRestore}}%
  {\global\contframefalse
   \endMakeFramed}
\makeatother
```

I admit that I have given little explanation of the code examples. The excuse that I would like you to believe is that adding all the explanatory material would make the article too long, but the real reason is that I do not really understand how the `framed` package performs its magic; hence the experiments (i.e., many trials and even more errors) that I did to create code that seemed workable.

References

[1] Donald Arseneau. The framed package v0.95, 2007. `mirror.ctan.org/macros/latex/contrib/framed`.

[2] Robert Nyqvist. 'example' environment or command. Post to `comp.text.tex` newsgroup, 11 January 2003.

[3] Peter Wilson. The memoir class for configurable typesetting, 2008. `mirror.ctan.org/macros/latex/contrib/memoir` or `mirror.ctan.org/install/macros/latex/contrib/memoir.tds.zip`.

[4] Peter Wilson. The changepage package, October 2009. `mirror.ctan.org/macros/latex/contrib/changepage`.

◇ Peter Wilson
20 Newfield Ave.
Kenilworth CV8 2AU, UK
herries dot press (at)
earthlink dot net

Some misunderstood or unknown LaTeX 2ε tricks (III)

Luca Merciadri

1 Introduction

After two other *TUGboat* articles, here is a third installment with more tips. We shall see:

1. how to print, in an easy way, a monochrome version of a document,
2. how to draw rightmost braces,
3. how to draw watermarks,
4. a plagiarism discussion and the related computational solutions.

2 Printing monochrome

When writing a 'screen-version' of some book, one often uses colors. However, if this screen-version needs to be printed in black and white, it is better to give it as a monochrome document. This can be achieved easily by simply adding `monochrome` to `color` and `xcolor`'s options. For example, if you called these packages without any options, it means that you might put

```
\usepackage[monochrome]{color}
\usepackage[monochrome]{xcolor}
```

in your preamble. Thanks to Enrico Gregorio [3] for this.

Herbert Voß gave me [3] a more technical PostScript version:

```
\AtBeginDocument{%
 \special{ps:
  /setcmykcolor {
    exch 0.11 mul add
    exch 0.59 mul add
    exch 0.3 mul add
    dup 1 gt { pop 1 } if neg 1 add setgray } def
  /setrgbcolor {
    0.11 mul
    exch 0.59 mul add
    exch 0.3 mul add setgray } def
  /sethsbcolor {
    /b exch def /s exch def 6 mul dup cvi dup
    /i exch def sub /f exch def
    /F [ [0 1 f sub 1] [f 0 1] [1 0 1 f sub]
         [1 f 0] [1 f sub 1 0] [0 1 f]
         [0 1 1] ] def
    F i get { s mul neg 1 add b mul} forall
    0.11 mul
    exch 0.59 mul add
    exch 0.3 mul add setgray } def
}}
```

Thanks to him too.

3 Drawing rightmost braces

It is common to synthesize some theorems' ideas by using a right brace, or simply to write such kinds of systems to define e.g. a function:

$$\left.\begin{array}{ll} -1 & x \leq 0 \\ 1 & x > 0 \end{array}\right\} \stackrel{\text{def}}{=} f(x). \tag{1}$$

This can be achieved by using various tricks, such as those which were proposed by Eduardo Kalinowski and Dan Luecking [2]:

- ```
  \left.
  \begin{array}
  ...\\
  ... \\
  ...
  \end{array}
  \right\}
  ```
- Use the `aligned`, `gathered`, or `alignedat` environments,

but one can also

- define a 'revert cases' environment, say `sesac`:

  ```
  \usepackage{amsmath}
  \makeatletter
  \newenvironment{sesac}{%
    \let\@ifnextchar\new@ifnextchar
    \left.%
    \def\arraystretch{1.2}%
    % One might prefer aligns other than left,
    % depending on the use to which it is put:
    \array{@{}l@{\quad}l@{}}%
  }{\endarray\right\}}
  \makeatother
  ```

  in the preamble, the `amsmath` package evidently being mandatory. One can then use `sesac` the way `cases` is used:

  ```
  \begin{equation}
   \begin{sesac}
     -1 & x \leq 0\\
      1 & x > 0
   \end{sesac} = f(x)
  \end{equation}
  ```

  Thanks to Dan Luecking for this [2]. The advantage of this definition is that its call is similar to the one which is used for the `cases` environment. Note that the `rcases` equivalent of `sesac` will be available in the `mathtools` package (from June, 2010), together with variants.

4 Using watermarks

There is sometimes a need for *watermarks*, either for security reasons, or simply for indicating important information along with the document.
  ```

There are basically three different ways to put watermarks in your LaTeX 2ε documents:

1. The `xwatermark` package,
2. The Ti*k*Z package,
3. The `draftcopy` package.

We shall discuss these different options separately now. We assume that the user wants to watermark all the pages. (Modifications to make only some pages be watermarked are easy.)

4.1 The `xwatermark` option

To watermark the *current* `tex` document, you can simply put [4]

```
\usepackage[printwatermark=true,
 allpages=true,fontfamily=pag,
 color=gray,grayness=0.9,
 mark=Draft,angle=45,
 fontsize=5cm,
 markwidth=\paperwidth,
 fontseries=b,scale=0.8,
 xcoord=0,ycoord=0]{xwatermark}
```

in your preamble, where parameters are modified in the obvious way.

4.2 The Ti*k*Z way

You can also use Ti*k*Z ([5]):

```
\begin{tikzpicture}[remember picture,overlay]
\node[rotate=0,scale=15,text opacity=0.1]
 at (current page.center) {Draft};
\end{tikzpicture}
```

writes the 'Draft' message in the center of the page, and

```
\begin{tikzpicture}[remember picture,overlay]
\node [xshift=1cm,yshift=1cm]
   at (current page.south west)
   [text width=7cm,fill=red!20,rounded corners,
    above right]
{
This is a draft!
};
\end{tikzpicture}
```

puts 'This is a draft!' in a box, at the desired place. Both might be put outside of the preamble. In both cases, you evidently need to load the `tikz` package. There are many other options (please check the package's manual).

The advantage of this approach is that you can call Ti*k*Z after or before some text to watermark the relative page, without knowing its number.

4.3 The `draftcopy` package

A third approach is to use the `draftcopy` package. You can then specify the intensity of the gray, the range of pages for which the word 'DRAFT' is printed and where it is printed (across the page or at the bottom). The package's feature are best described in its manual [6], but, roughly,

```
\usepackage[english,all,
 portrait,draft]{draftcopy}
```

should suit your needs.

5 LaTeX 2ε and plagiarism

Plagiarism is a well-known issue. It is defined as (*Random House Compact Unabridged Dictionary*, 1995)

> The use or close imitation of the language and thoughts of another author and the representation of them as one's own original work.

I will here take a very concrete case: my own. This year, I had a group project to do, and the two other students did not contribute at all. As I had to share my work with them because there was an oral exam and that the professor wanted it to be shared, I accepted to share it, but with a big watermark.

I had not realized that this choice would be critical. Some days later, I realized that one of the two other students wanted to appropriate the work, and thereby claim its honesty and investment in the work. He tried to remove the watermark, but, despite much research, never found out how. However, he could have done it. I learnt many things thanks to this situation, which I will explain here from a TeX point of view.

My first reaction to ensure security was to *secure the PDF by using free tools*. This was a good dissuasion, as the two other students were stopped by this measure. By 'secure', I mean that I theoretically prevented others from printing, selecting, or extracting content from the PDF file. However, such PDF 'security' is not widely accepted by PDF readers. Here is what Jay Berkenbilt (`qpdf`'s author) once told me [1]:

> The PDF specification allows authors of PDF files to place various restrictions on what you can do with them. These include restricting printing, extracting text and images, reorganizing them, saving form data, or doing various other operations. These flags are nothing more than just a checklist of allowed operations. The PDF consuming application (`evince`, Adobe Reader, etc.) is supposed to honor those restrictions and prevent you from doing operations that the author didn't want you to do.
>
> The PDF specification also provides a mechanism for creating encrypted files. When a PDF file is encrypted, all the strings and stream

> data (such as page content) are encrypted with a specific encryption key. This makes it impossible to extract data from without understanding the PDF encryption methods. (You couldn't open it in a text editor and dig for recognizable strings, for example.) The whole encryption method is documented in the specifications and is basically just RC4 for PDF version 1.4 and earlier, which is not a particularly strong encryption method. PDF version 1.5 added 128-bit AESv2 with CBC, which is somewhat stronger. Those details aren't really important though. The point is that you must be able to recover the encryption key to decrypt the file.
>
> Encrypted PDF files always have both a user password and an owner password, either or both of which may be the empty string. The key used to encrypt the data in the PDF is always based on the user password. The owner password is not used at all. In fact, the only thing the owner password can do is to recover the user password. In other words, the user password is stored in the file encrypted by the owner password, and the encryption key is stored in the file encrypted by the user password. That means that it is possible to entirely decrypt a PDF file, and therefore to bypass any restrictions placed on that file, by knowing only the user password. PDF readers are supposed to only allow you to bypass the restrictions if you can correctly supply the owner password, but there's nothing inherent [in] the way PDF files are structured that makes this necessary.
>
> If the user password is set to a non-empty value, neither `qpdf` nor any other application can do anything with the PDF file unless that password is provided. This is because the data in the PDF file is simply not recoverable by any method short of using some kind of brute force attack to discover the encryption key.
>
> The catch is that you can't set restrictions on a PDF file without also encrypting it. This is just because of how the restrictions are stored in the PDF file. (The restrictions are stored with the encryption parameters and are used in the computation of the key from the password.) So if an author wants to place restrictions on a file but still allow anyone to read the file, the author assigns an empty user password and a non-empty owner password. PDF applications are supposed to try the empty string to see if it works as a password, and if it does, not to prompt for a password. *In this case, however, it is up to the application to voluntarily honor any of the restrictions imposed on the file.* [italics mine —lm] This is pretty much unavoidable: the application must be able to fully decrypt the file in order to display it.
>
> None of this is a secret. It's all spelled out in the PDF specification. So encrypting a PDF file without a password is just like encrypting anything else without a password. It may prevent a casual user from doing something with the data, but there's no real security there. Encrypting a PDF file with a password provides pretty good security, but the best security would be provided by just encrypting the file in some other way not related to PDF encryption.

Thus, one might not want to rely only on this PDF feature, especially if the desire is to set attributes without a password (see the slanted sentence in the cited text).

My second idea was to *put a watermark on every page of the document*. For this, I used the `xwatermark` package, because I had no time to look for another way to achieve it (I was near the work's due date).

I then compiled the `tex` document, secured it, and sent it.

In this series of practices, I should have realized that these two protections could totally be circumvented in an easy way. I knew it, partially, but had not much time to think about it. One needs to realize that such practices are not infallible: they are only ways to discourage, not absolutely prevent, your work from being plagiarized.

Let's take, for example, the PDF security. One could simply run `pdf2ps`, and then `ps2pdf` on the resulting PostScript file, to recover exactly the same PDF without the security attributes (or hyperlinks). Thus, by using two commands, you can easily remove the PDF protection (assuming it was only concerning attributes, not passwords).

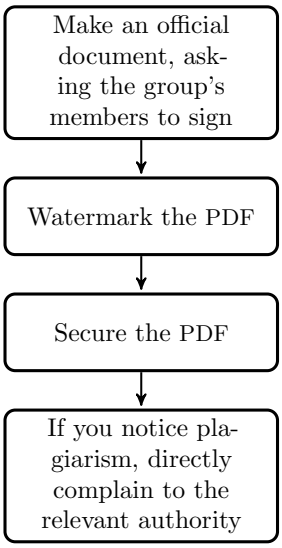
Next, there is the watermark that was LaTeX 2_ε-inserted. There are different programs, especially for Windows users, that can remove watermarks. I tried them on my watermark, and none could remove it. Good point, but that does not mean that there is no program which is able to remove it. I might have forgotten one, or simply, a commercial program might be capable of this. (I never test commercial programs.) But I had made an important mistake in my LaTeX 2_ε watermark. The watermark is written

on a different PDF 'layer' (one can conceive of a PDF as being constructed from different layers which are superimposed in some way) and is thereby not completely incorporated in the core document. Thus, if you use a LaTeX 2_ε package to write a watermark in a document, do not forget to *merge layers*. This can be achieved easily. For example, using `lpr` under GNU/Linux, you can re-print the original PDF document (with the different PDF layers) as another PDF where the watermark and the text layers are merged together, thus making differentiating between them very complicated for a 'normal' user. This can be achieved with a GUI too, evidently. For me, I can directly choose this virtual printer, and to print to a file, through GNOME's printing interface.

But one needs to keep in mind that all these measures are here only as a method for discouraging. For example, whatever the protection, one can still take screenshots of the PDF viewer's window, to make copies of the PDF content. This is tedious, but if one wants to, it can be done. If even these screenshots were somehow made impossible, he could use a camera to take pictures of his screen. All the measures you need to take against such behavior need to be adapted, and correlated in regards to other's motivation to exploit your work. This is a very important point, as, once the work is sent, you cannot modify what you gave.

Another point which could be exploited is the use of a somewhat legal document, constraining the group's members to sign.

The best thing is presumably to avoid these problems by talking together, as we humans are equipped with an extraordinary ability to share their feelings and to express themselves; however, communication problems sometimes arise, and, in this case, you might think about the aforementioned tricks. Here is thus what I suggest you to do, if such a problem arises (the first arrow being to follow if communication is somewhat broken):



◇ Luca Merciadri
University of Liège
`Luca.Merciadri (at) student dot ulg dot ac dot be`
`http://www.student.montefiore.ulg.ac.be/~merciadri/`

References

[1] Berkenbilt, Jay. (PDF specification message), 2010. `http://www.mail-archive.com/debian-user@lists.debian.org/msg570956.html`.

[2] Merciadri, Luca, Kalinowski, Eduardo and Luecking, Dan. 'cases' environment for a brace in the other sense? (`comp.text.tex` discussion), 2010.

[3] Merciadri, Luca, Voß, Herbert and Gregorio, Enrico. dvi, ps, pdf in black and white: how to do it if I have colors? (`comp.text.tex` discussion), 2010.

[4] Musa, Ahmed. The xwatermark Package, 2010. `http://mirror.ctan.org/macros/latex/contrib/xwatermark/xwatermark-guide.pdf`.

[5] Tantau, Till. Ti*k*Z, PGF, 2008. `http://mirror.ctan.org/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf`.

[6] Vollmer, Jürgen. The draftcopy package, 2006. `http://www.ifi.uio.no/it/latex-links/draftcopy.pdf`.

LaTeX3 News

Issue 5, January 2011

Happy new year

Seasons greetings for 2011! As the previous news issue was released late, this season's issue will be shorter than usual.

The LPPL is now OSI-approved

We are happy to report that earlier this year the LaTeX Project Public License (LPPL) has been approved by the OSI as an open source licence.[1] Frank Mittelbach has further details on this news in a retrospective of the LPPL [in this *TUGboat*].

Reflections on 2010

We are pleased to see the continued development and discussion in the TeX world. The LaTeX ecosystem continues to see new developments and a selection of notable news from the second half of last year include:

June The TUG 2010 conference was held very successfully in San Francisco; videos, slides, and papers from LaTeX3 Project members are available from our website.[2]

Aug. The TeX Stack Exchange[3] question & answer website was created and has since grown quickly. At time of writing, some 2800 people have asked 2600 questions with 5600 answers total, and 2200 users are currently visiting daily.

Sept. TeX Live 2010 was released: each year the shipping date is earlier; the production process is becoming more streamlined and we congratulate all involved for their hard work. One of the most notable new components of TeX Live 2010 includes the 'restricted shell escape' feature to allow, among other things, automatic EPS figure conversion for pdfLaTeX documents.

Oct. TLContrib[4] was opened by Taco Hoekwater as a way to update a TeX Live installation with material that is not distributable through `tlmgr` itself. Such material includes executables (e.g., new versions of LuaTeX), non-free code, or test versions of packages.

Nov. Philipp Lehman released the first stable version of biblatex. One of the most ambitious LaTeX packages in recent memory, biblatex is a highly flexible package for managing citation cross-referencing and bibliography typesetting. In 'beta' status for some years now, reaching this point is a great milestone.

Dec. LuaTeX 0.65. We are happy to see LuaTeX development steadily continuing. LaTeX users may use LuaTeX with the `lualatex` program. Like `xelatex`, this allows LaTeX documents to use multilingual OpenType fonts and Unicode text input.

Current progress

The expl3 programming modules continue to see revision and expansion; we have added a LuaTeX module, but expl3 continues to support all three of pdfLaTeX, XeLaTeX, and LuaLaTeX equally.

The l3fp module for performing floating-point arithmetic has been extended and improved. Floating point maths is important for some of the calculations required for complex box typesetting performed in the new 'coffins' code. The l3coffin module has been added based on the original xcoffins package introduced at TUG 2010 as reported in the last news issue; this code is now available from CTAN for testing and feedback.

We have consolidated the l3int and l3intexpr modules (which were separate for historical purposes); all integer/count-related functions are now contained within the 'int' code and have prefix `\int_`. Backwards compatibility is provided for, but eventually we will drop support for the older `\intexpr_` function names.

Plans for 2011

In the following year, we plan to use the current LaTeX3 infrastructure to continue work in building high-level code for designing LaTeX documents using the xtemplate package. Our first priority is to look at section headings and document divisions, as we see this area as one of the most difficult, design-wise, of the areas to address. From there we will broaden our scope to more document elements.

We will also do some low-level work on the 'galley', which is the code that LaTeX3 uses to build material for constructing pages, and we will continue to extend expl3 into a more complete system from which we can, one day, create a pure LaTeX3 format.

[1] `http://www.opensource.org/licenses/lppl`
[2] `http://www.latex-project.org/papers/`
[3] `http://tex.stackexchange.com`
[4] `http://tlcontrib.metatex.org/`

Book review: *Typesetting tables with LaTeX*

Boris Veytsman

Herbert Voß, *Typesetting tables with LaTeX*. UIT Cambridge, 2011. 231 pp. Paperback, US$29.95

More than fifteen years ago my friend defended a doctoral thesis at Stanford. He wrote it in plain TeX; everything went fine except tables. My friend spent some time trying to align the numbers exactly like he wanted them, but failing to do so, he eventually just printed the table as plain text and pasted it into the manuscript.

Proper typesetting of tables is one of the most difficult and complex TeX tasks. LaTeX facilities for typesetting tabular material are arguably easier to use than the ones of plain TeX. Still, when one is not satisfied with the default look and feel of LaTeX tables (as many typographers are not), then one may spend long hours making tables look right. Fortunately, there are two scores of packages on CTAN dealing with adjusting typesetting of tables, so one can find code dealing with the specific task at hand. Unfortunately, there *are* two scores of packages on CTAN dealing with adjusting typesetting of tables, so it is rather difficult to find the one needed. The indispensable *Companion* describes some packages and tricks, but the material is clearly much larger than can be fitted in a chapter of the general book. Therefore there is a clear need for a specialized review of table-related packages.

The new book by Herbert Voß is intended to be such a review. It covers both the standard LaTeX facilities for dealing with tabular material, and 39 packages dealing with tables that are currently on CTAN. Each package is discussed concisely but clearly. The reader gets an impression of the main features of the package and the basic usage, being referred to the package documentation for the details of the usage and the less popular features. This design of the presentation is a very good idea: the information in the book is enough to find the package for the given need without swelling the book volume with too much detail. Besides, the details of interface and minor features tend to change from release to release, so a printed book that includes too much detail quickly becomes obsolete.

From this book the reader can find out how to change the width and height of the table cells, how to make the tables look more "professional" than the default LaTeX tables, how to get multirow and multicolumn cells, how to use footnotes with the tables, how to get colored cells and lines, how to typeset the numbers in tables right, how to get "spreadsheet-like" arithmetic in tables, how to get rotated and multipage tables, how to read comma separated values files into TeX — and much more.

The author spends considerable time discussing interaction between packages — an important topic often omitted in the documentation. Each package is accompanied by carefully chosen examples of its usage. There are also separate sections on *Tips and Tricks* and *Examples*. The gallery of examples is very impressive and may provide inspiration for many beautiful tables.

The book is well written and clear. It should be easy reading for a novice — and still it provides many insights for an advanced TeXnician.

The book is very nicely typeset and designed. UIT Cambridge is known for the good taste in its production, and this book is no exception. (I was surprised, however, that while the cover had "LaTeX", the spine typesetting was "LATEX".)

Of course any book, including this one, can be made better. It seems to me that the largely alphabetical ordering of packages in this book can be improved. Right now there are three major divisions of packages: general, color-related packages and multipage tables. Probably one can find some logical subdivisions in the general category. Further, while the book clearly explains *how* to change the typesetting of tables, it lacks the discussion of *why* to do this and even *whether* to do some of them (like rotated table headers). The author does mention some typographic rules with respect to tabular material (like not using vertical lines, and spare use of horizontal ones). Nevertheless a separate section on *Fundamentals of Typography of Tables* would be very useful. Finally, a discussion of typesetting table captions would probably be a good addition to the book. That standard LaTeX puts the caption *below* the table is an unfortunate violation of the traditions of good typography. There are a number of packages that correct this mistake and provide many options of customization of captions.

However, it is probably impossible to write a book that would satisfy everyone's wishlist (and such a book would be too heavy to carry around). *Typesetting tables with LaTeX* by Herbert Voß is a good reference on LaTeX packages related to tabular material. It should be a necessary addition to the bookshelf of a LaTeX user.

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

The Treasure Chest

This is a list of selected new packages posted to CTAN (`http://www.ctan.org`) from October 2010 through March 2011, with descriptions based on the announcements and edited for brevity.

Entries are listed alphabetically within CTAN directories. A few entries which the editors subjectively believed 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
`http://tug.org/ctan.html`

fonts

`bbold-type1` in `fonts`
Type 1 versions of the `bbold` fonts.

`comfortaa` in `fonts`
Support for the Comfortaa sans serif family designed by Johan Aakerlund.

* `droid` in `fonts`
Support for the Droid family made for Android.

`lato` in `fonts`
Support for the Lato sans serif family designed by Łukasz Dziedzic.

`mathstone` in `fonts`
Use Adobe Stone Sans and Stone Serif for math.

`ocr-b-outline` in `fonts`
OCR-B fonts in Type 1 and OpenType, compatible with the existing Metafont metrics.

`pcarl` in `fonts`
Support for Adobe Caslon Open Face.

`prodint` in `macros/latex/contrib`
Product integral notation.

`ptsans` in `fonts`
Support for the ParaType PT Sans family.

`ptserif` in `fonts`
Support for the ParaType PT Serif family.

`rsfso` in `fonts`
Support for using `rsfs` as math calligraphic.

`urwchancal` in `fonts`
Support for using Zapf Chancery as a math alphabet.

`verdana` in `fonts`
Support for Verdana.

`fonts/bbold-type1`

graphics

`pgfgantt` in `graphics/pgf/contrib`
Ti*k*Z package for drawing Gantt charts.

`pst-graphicx` in `graphics/pstricks/contrib`
Support using LaTeX's `graphicx` in plain TeX.

`pst-tvz` in `graphics/pstricks/contrib`
PSTricks package for drawing trees with more than one root node.

`threeddice` in `graphics/metapost/contrib/macros`
MetaPost package for drawing dice.

info

`LaTeX-Bib` in `info/examples`
Examples from the book *Bibliografien mit LaTeX*.

* `lualatex-doc` in `info/luatex`
Guide to the world of LuaLaTeX, plus information about Plain LuaTeX.

`tabulars-e` in `info/examples`
Examples from the book *Typesetting tables with LaTeX*.

language

`spanglish` in `language/spanish/babel/contrib`
Simplified and sloppy Spanish support (for when Babel's `spanish.ldf` fails).

macros/generic

`fntproof` in `macros/generic`
Font proofing as in `testfont.tex`.

`navigator` in `macros/generic`
PDF features across formats and engines.

`systeme` in `macros/generic`
Vertical alignment of systems of equations.

macros/latex/contrib

`acroterm` in `macros/latex/contrib`
Manage and index acronyms and terms.

`adjmulticol` in `macros/latex/contrib`
Adjusting margins for multi- and single-column output.

`aomart` in `macros/latex/contrib`
Typesetting articles for *The Annals of Mathematics*.

`apa6e` in `macros/latex/contrib`
Formatting according to the American Psychological Association 6th edition guidelines.

`babeltools` in `macros/latex/contrib`
Flexibility on top of `babel`.

`bashful` in `macros/latex/contrib`
Support execution of Bash scripts from within LaTeX.

`bibleref-french` in `macros/latex/contrib`
French adaptation of `bibleref`.

bibleref-german in macros/latex/contrib
German adaptation of bibleref.

bibleref-parse in macros/latex/contrib
Parse Bible passages given in a wide variety of human-readable formats.

bondgraph in macros/latex/contrib
Draw bond graphs using PGF/TikZ.

booktabs-de in macros/latex/contrib
German translation of the booktabs documentation.

canoniclayout in macros/latex/contrib
memoir extension for a page layout based on circles.

catchfilebetweentags in macros/latex/contrib
Extract a portion of a file between tags into a macro.

catoptions in macros/latex/contrib
Preserve and recall standard catcodes.

chemexec in macros/latex/contrib
Prepare exercise sheets with separate solutions.

collcell in macros/latex/contrib
Collect content of a tabular cell to pass to a macro.

colourchange in macros/latex/contrib
Change colors of structural elements in Beamer during a presentation.

* cprotect in macros/latex/contrib
Allow any control sequence (\footnote, ...) to take verbatim text in its argument.

dfgproposal in macros/latex/contrib
Proposals for the German Research Council (DFG).

dirtytalk in macros/latex/contrib
Typeset (possibly nested) quoted text.

easy-todo in macros/latex/contrib
TODO notes in the document body and as an index.

enumitem-zref in macros/latex/contrib
General references for items in enumitem lists.

fileinfo in macros/latex/contrib
Standalone and expandable \GetFileInfo.

filemod in macros/latex/contrib
Read and compare file modification times.

finstrut in macros/latex/contrib
Fix \@finalstrut to not produce an empty line in vertical mode.

fldigigal in macros/latex/contrib
Create an AcroTEX Flash Digital Gallery.

geometry-de in macros/latex/contrib
German translation of the geometry documentation.

gincltex in macros/latex/contrib
Support including external LaTeX files as graphics.

gmp in macros/latex/contrib
Integrate MetaPost and LaTeX.

gradientframe in macros/latex/contrib
Produce gradient frames around objects.

* hardwrap in macros/latex/contrib
Break lines by words to a specified width.

he-le-na in macros/latex/contrib
Explicit hyphenations and shortcuts for Serbian.

he-she in macros/latex/contrib
Alternate masculine and feminine pronouns.

* interfaces in macros/latex/contrib
Key–value interface for frequently-changed settings provided by several packages.

iwhdp in macros/latex/contrib
Format discussion papers of the Halle Institute for Economic Research.

koma-moderncvclassic in macros/latex/contrib
Make moderncv (classic style) available for koma classes and thus compatible with biblatex.

libgreek in macros/latex/contrib
Libertine/Biolinum Greek in math mode.

lpic in macros/latex/contrib
Typeset LaTeX material over included graphics.

mathastext in macros/latex/contrib
Propagate document text font to math mode.

morehype in macros/latex/contrib
Shorthands for TeX-related hyperlinks; improved tables of contents with hyperref; generating HTML with TeX macros.

mpgraphics in macros/latex/contrib
Write MetaPost diagrams inline; process with a single run.

msuthesis in macros/latex/contrib
Typeset Michigan State University graduate theses.

musixguit in macros/latex/contrib
Simplify guitar notation with musixtex.

mychemistry in macros/latex/contrib
Typeset complex chemical reaction schemes.

parselines in macros/latex/contrib
Line-by-line for files or environment contents.

physymb in macros/latex/contrib
Shortcuts for physicists.

piano in macros/latex/contrib
Draw two-octave piano keyboard with up to seven selected keys highlighted.

productbox in macros/latex/contrib
Typeset a three-dimensional product box.

pxgreeks in macros/latex/contrib
Select italic or upright shapes for the pxfonts Greek.

randomwalk in macros/latex/contrib
Draw random walks.

rec-thy in macros/latex/contrib
Typesetting recursion (computability) theory papers.

tabu in macros/latex/contrib
Flexible LaTeX tabulars, enhancing tabular, array, and tabularx.

textgreek in macros/latex/contrib
Upright greek letters as text, e.g., \textbeta.

tkz-base in macros/latex/contrib/tkz
Base package for easier programming with TikZ.

tkz-euclide in macros/latex/contrib/tkz
Euclidean geometry drawings with tkz-base.

tkz-fct in macros/latex/contrib/tkz
Graphs of functions with tkz-base and Gnuplot.

`tucv` in `macros/latex/contrib`
Typeset a resume or CV.

`turnthepage` in `macros/latex/contrib`
Indicate on two-sided documents to turn the page.

`txgreeks` in `macros/latex/contrib`
Select italic or upright shapes for the `txfonts` Greek.

`uothesis` in `macros/latex/contrib`
Typeset University of Oregon theses.

`uri` in `macros/latex/contrib`
Support for DOI, XMPP, and many other types of URIs.

`ytableau` in `macros/latex/contrib`
Many-featured Young tableaux and Young diagrams.

macros/latex/contrib/beamer-contrib

`beamer2thesis` in `m/l/c/beamer-contrib/themes`
Beamer theme for thesis presentations.

`nirma` in `m/l/c/beamer-contrib/themes`
Beamer theme for academic presentations.

macros/latex/contrib/biblatex-contrib

`biblatex-ieee` in `m/l/c/biblatex-contrib`
`biblatex` support for the IEEE style.

macros/luatex

`luasseq` in `macros/luatex/generic`
Drawing spectral sequences in LuaLaTeX.

`luaindex` in `macros/luatex/latex`
Index processor in Lua.

macros/xetex

`fixlatvian` in `macros/xetex/latex`
Extend `polyglossia`'s Latvian language support.

`*ucharclasses` in `macros/xetex/latex`
Code insertion between characters from different Unicode blocks.

`unisugar` in `macros/xetex/latex`
Unicode characters used in common LaTeX commands, and support control sequence names with right-to-left characters.

support

`purifyeps` in `support`
Convert EPS files to a form readable to pdfTeX, using `pstoedit`.

`sty2dtx` in `support`
Initialize a `.dtx` file from a `.sty`.

`texdef` in `support`
Display definition of a given (La)TeX control sequence.

‘Magic’ comments in TEXworks 0.4

Joseph Wright

The editor TEXworks (`http://tug.org/texworks`) is designed to ‘lower the entry barrier to the TEX world’. TEXworks v0.4 has recently been released, and this is a good opportunity to look at one very useful feature: ‘magic’ comments. These are used to give the editor information about the file being edited. (The concept is not unique to TEXworks: TeXShop and AUCTEX both include similar ideas.)

```
% !TeX program = LuaLaTeX
```

specifies the name of the typesetting engine to use for the current file, which should be one of the engines that is set up for use with TEXworks. This is useful if you normally use one engine (for example `pdfLaTeX`), but have a few files that need an alternative engine. In the example, the file would automatically be processed with `LuaLaTeX` as the engine.

```
% !TeX encoding = UTF-8
```

Sets the file encoding for the current file. The usual default is `UTF-8`, but this setting is handy if you need to collaborate with other people using non-UTF-8 editors.

```
% !TeX root = somefile.tex
```

Indicates that the current file is not the main file for typesetting: when you choose to typeset, TEXworks will save the current file then typeset the master file. Using this setting, you need the full name of the master file including the extension. This is clearly a handy setting for larger projects, where you might have a lot of files which are to be included in one master document.

```
% !TeX spellcheck = de-DE
```

Specifies the spell check language for the current file. The language of course needs to be one you have installed: TEXworks currently uses the hunspell dictionary format. OpenOffice.org also uses this format, but has recently extended it. More information is at `http://code.google.com/p/texworks/wiki/SpellingDictionaries`.

One point to notice with the `root` setting is how it interacts with `program`. Let’s imagine that the master file needs to be typeset using `LuaLaTeX`, and that your default engine is `pdfLaTeX`. You then need to include the program in each subsidiary file to get everything to work properly:

```
% !TeX root    = master.tex
% !TeX program = LuaLaTeX
```

Without this, when you try to typeset from one of the subfiles then TEXworks will use the currently-selected engine (probably `pdfLaTeX`) and not `LuaLaTeX` for the typesetting. Once you know, this is not so surprising, but at first it is easy to get caught out!

TEXworks 0.4 includes plenty of other new features and bug fixes. Perhaps the most notable is scripting support for QtScript, Lua and Python, including now-bundled scripts for some common tasks. See the web page at `http://tug.org/texworks` for more information and a full list of changes.

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

***Eutypon* 24–25, October 2010**

Eutypon is the journal of the Greek TEX Friends (`http://www.eutypon.gr`).

Kiki Dimoula, The audacious word thief; p. 1
(A poem.) (*In Greek.*)

John Plaice, On Omega (Ω) and beyond; pp. 3–9
(An interview, conducted by Apostolos Syropoulos.) John Plaice is known for his pioneering work on Ω (Omega), the first project to expand the multilingual capabilities of TEX in the early and mid-1990s. That project is now over, but as John explains in this interview, its heritage is still alive in LuaTEX, and also in XML. John also talks about Cartesian Programming, his new project that one day may bring to life the *Cartesian document*. (*Article in English.*)

Apostolos Syropoulos, Uniform rendering of XML encoded mathematical content with OpenType fonts; pp. 11–22

The new OpenType MATH font table contains important information that can be used to correctly and uniformly render mathematical content (e.g., mathematical formulae, equations, etc.). Until now, all systems rendering XML encoded mathematical content employed some standard algorithms together with some standard sets of TrueType and/or Type 1 fonts, which contained the necessary glyphs. Unfortunately, this approach does not produce uniform results because certain parameters (e.g., the thickness of the fraction line, the scale factor of superscripts/subscripts, etc.) are system-dependent, that is, their exact values will depend on the particular setup of a given system. Fortunately, the new OpenType MATH table can be used to remedy this situation. In particular, by extending renderers so as to be able to render mathematical contents with user-specified fonts, the result will be uniform across systems and platforms. In other words, the proposed technology would allow mathematical content to be rendered the same way ordinary text is rendered across platforms and systems. (*Article in English.*)

Ioannis A. Vamvakas, An improved version of "Byzantine" music fonts; pp. 23–40

In this paper, we present a second, revised version of our "Byzantine" music fonts. We also present a new approach for a more efficient use of these fonts with LATEX, and its ancestor TEX. (*Article in Greek with English abstract.*)

Alexandros Droseltis, GNU LilyPond: a music engraving program; pp. 41–57

In this article the music engraving program GNU LilyPond is presented. At first, the basic commands of the program are presented, which control pitch, durations, dynamics, expression and articulation signs, lyrics and various possibilities of typesetting polyphony, and the use of variables for the sake of code simplification. Further, the two most important concepts of organizing the score are explained, contexts and engravers, and an introduction is made to the basic commands that change the defaults. At the end, the most compatible mode of integrating music scores in text files is mentioned, as well as some auxiliary applications, the documentation of the program and the support of the users. (*Article in Greek with English abstract.*)

Triantafyllos E. Sklavenitis, The typographer Christos G. Manousaridis, 1936–2008; pp. 59–65

The Greek master typographer Christos G. Manousaridis passed away at the beginning of 2008. His print shop, where some of the most difficult and magnificent Greek documents were produced in the latter half of the 20th century, had already ceased its operation since 2007. With the passing of Manousaridis, the era of metal type typography seems to end forever in Greece. Nonetheless, the products of his art will remain as lessons of perfection and aestheticism for the new generations of Greek typographers. (*Article in Greek with English abstract.*)

[Received from Dimitrios Filippou
and Apostolos Syropoulos.]

MAPS 41 (2010)

MAPS is the publication of NTG, the Dutch language TEX user group (`http://www.ntg.nl`).

[Received from Taco Hoekwater.]

***The PracTEX Journal* 2010-2**

The PracTEX Journal is an online publication of the TEX Users Group. Its web site is `http://tug.org/pracjourn`. All articles are available there.

Issue theme: LATEX for teachers.

Lance Carnes, In this issue
Editor's introduction to the issue.

The Editors, News from Around
Knuth's Earthshaking Announcement; More Knuth humor; Type maps.

Cassiano S. Rosa and Og DeSouza, Sweave — Interface entre R e LATEX [Using R and LATEX]
When using R for statistical analyses, it is common to keep the data analyses, the results of experiments, and graphs in separate files. Fortunately, for R users who also use LATEX, there is a tool for organizing these files: Sweave! This paper presents a very short account on how Sweave integrates R and LATEX to keep both input and output of statistical analyses in a single style file. (In Portuguese.)

Alain Schremmer, Configurable materials for teaching mathematics
This article describes a system that uses LATEX to generate math texts, homework, quizzes, and exams for developmental mathematics courses.

Marius Hofert and Markus Kohm, Scientific presentations with LATEX
In this article, we show how scientific presentations can be created based on the KOMA-Script document class `scrartcl`. The main advantage of the suggested approach is that the presentation slides allow for easy copy-and-paste of content from other LATEX documents such as research papers or handouts. Using this approach, presentation slides are quickly generated, without the author having to learn how to use other LATEX presentation packages. Additionally, in contrast to the rather overused templates of the more common presentation packages, the slides can be individually created and thus tailored to the topic of the presentation.

Paulo Rogério and Rian Gabriel, Design and preparation of effective scientific posters using LATEX
Posters are important presentation tools in scientific conferences. LATEX offers several packages, e.g. `a0poster` and `sciposter`, for designing several kinds of high quality posters. However, many of the posters we are used to seeing are visually split into columns and conceptually organized in sections, with amounts of text which are likely to disrupt the viewing experience and understanding of the content. In this article we present an efficient method for preparing visual scientific posters using the PGF package and its syntax layer Ti*k*Z.

Cristina Blaga and Paul Blaga, Variation and sign tables with tableau
We describe here a package, `tableau.sty`, created by N. Kisselhoff, very useful especially for calculus courses. It aids in the construction of variation and sign tables for the study of functions. The package provides a new environment based on PSTricks.

Cristina Blaga and Paul Blaga, Preparing exam and homework material with `probsoln`
We describe here some of the possibilities provided by the package `probsoln`, by Nicola Talbot. The aim of the package is to help the user prepare different kinds of problem lists and tests.

Bastiaan Jacques, Square, multiline cells using tabular(x)
I describe a method for creating square cells, containing multiple lines typeset in paragraph mode using the `array` package. Both plain LATEX `tabular` and `tabularx` packages are used.

The Editors, Ask Nelly
Footnotes appear above a figure?; Changing margins and line spacing?.

The Editors, Distraction: KenKen puzzles

Die TEXnische Komödie 2010/4–2011/1

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`http://www.dante.de`). [Editorial items are omitted.]

Die TEXnische Komödie 4/2010

Christine Römer, Gewichten Wichtiges und Unwichtiges mit LATEX markieren [Emphasizing text — Marking important and unimportant with LATEX; Part 1: Footnotes]; pp. 22–35

Among other things typography provides means for controlling the processing of information for the reader. Part of this is the establishment of certain patterns to indicate more important or less important facts in text. Some of them are discussed in this article. In the first part we cover functions and adjustment parameters of footnotes. In the second part various ways of highlighting text will be discussed.

Uwe Ziegenhagen, Datenanalyse mit Sweave, LATEX und R [Data analysis with R/Sweave and LATEX]; pp. 35–45

[Translation of the article in *TUGboat* 31:2.]

Rolf Niepraschk, Mehrere Stichwortverzeichnisse im LATEX-Dokument [Multiple indexes in a LATEX document]; pp. 46–50

In extensive documents it may make sense to list certain terms in the appendix. In this article we will show how to create these lists, using registers for persons and places as examples. It is not the aim of this article to describe all aspects of this topic in detail but rather to provide hints and ideas for dealing with it.

Herbert Voss, Das Paket `cutwin` [The `cutwin` package]; pp. 51–55

The `cutwin` package with its macros and environments allows cutting out pieces of text as a "window" in a paragraph if it contains only text. The macros are based on code first published by Alan Hoenig; further adjustments by Peter Wilson simplified the usage.

Die TEXnische Komödie 1/2011

Christine Römer, Gewichten Teil 2: Auszeichnungen [Emphasizing text — Marking important and unimportant with LATEX; Part 2: Emphases]; pp. 7–16

Emphases are mainly used to control the reading fluency. The intensity, by which the different typographic means of emphasizing text are used, depend not only on the kind of text, the targeted audience and the purpose but also on the "zeitgeist". Therefore general rules, which do not take the mentioned aspects into consideration, are of little help.

LATEX allows the use of all kinds of emphasizing text, however, there may be restrictions with some fonts which do not have the full character set.

Günter Rau, SageTEX; pp. 17–21

The software introduced here is a mathematical software system consisting of nearly 100 open source components accessible via a common Python interface.

With Sage (`http://www.sagemath.org`) there is a platform-independent LATEX package to include results directly into LATEX. There is also an online version which may be accessed via `http://www.sagenb.org`.

The examples in this article were created using version 4.6 under Debian Lenny.

Patrick Ossmann, Kyrillische Schriftzeichen im LATEX-Dokument [Cyrillic characters in LATEX documents]; pp. 22–29

In the western European language area Latin characters are used. Most of the time these characters are sufficient to handle everyday requirements, but when dealing with complex mathematical topics one may reach the limits of Latin and Greek characters. In these situations it is necessary to include Cyrillic characters in a LATEX document. At first glance this may seem trivial with LATEX but actually it is not!

The article deals with this topic and aims to provide a guideline on the use of Cyrillic characters in documents which are encoded in T1.

Kurt Lidwin, Ein passendes Bewerbungsanschreiben zum ModernCV-Lebenslauf [Suitable application letters for the ModernCV package]; pp. 30–39

In the various LATEX discussion groups and forums a question is raised from time to time: How can one create a nice application letter for a CV that was created with ModernCV?

This article provides a tutorial on how to create a matching layout application letter written with `scrlttr2` for a ModernCV curriculum vitae.

Herbert Voss, Einlesen und Ausführen von Quellcode [Displaying and executing source code]; pp. 40–54

A common question on mailing lists and in forums is if not only literal source code can be embedded in LATEX documents but also the results created by this code.

Packages such as `fancyvrb` and `listings` support external writing and partial reading of code. Further packages such as `showexpl` also provide means to execute the embedded code externally. In this article it is shown how arbitrary code can be treated this way.

[Received from Herbert Voß.]

ArsTEXnica #10 (October 2010)

ArsTEXnica is the journal of GUIT, the Italian TEX user group (`http://www.guit.sasup.it/`).

[Received from Gianluca Pignalberi.]

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.

Hendrickson, Amy
Brookline, MA, USA
Email: `amyh (at) texnology.com`
Web: `http://www.texnology.com`
LaTeX macro writing our speciality for more than 25 years: macro packages for major publishing companies, author support; journal macros for American Geophysical Union, Proceedings of the National Academy of Sciences, and many more.

Hendrickson, Amy (cont'd)
Scientific journal design/production/hosting, e-publishing in PDF or HTML.

LaTeX training, at MIT, Harvard, many more venues. Customized on site training available.

Please visit our site for samples, and get in touch. We are particularly glad to take on adventurous new uses for LaTeX, for instance, web based report generation including graphics, for bioinformatics or other applications.

Latchman, David
4113 Planz Road Apt. C
Bakersfield, CA 93309-5935
+1 518-951-8786
Email: `dlatchman (at) gmail.com`
Web: `http://www.elance.com/s/dlatchman`
Proficient and experienced LaTeX typesetter for books, monographs, journals and papers allowing your documents and books to look their possible best especially with regards to technical documents. Graphics/data rendered either using Ti*k*Z or Gnuplot. Portfolio available on request.

Peter, Steve
295 N Bridge St.
Somerville, NJ 08876
+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.

Shanmugam, R.
No. 38/1 (New No. 65), Veerapandian Nagar, Ist St.
Choolaimedu, Chennai-600094, Tamilnadu, India
+91 9841061058
Email: `rshanmugam92 (at) yahoo.com`
As a Consultant, I provide consultation, training, and full service support to individuals, authors, typesetters, publishers, organizations, institutions, etc. I support leading BPO/KPO/ITES/Publishing companies in implementing latest technologies with high level automation in the field of Typesetting/Prepress, ePublishing, XML2PAGE, WEBTechnology, DataConversion, Digitization, Cross-media publishing, etc., with highly competitive prices. I provide consultation in building business models &

Shanmugan, R. (cont'd)
technology to develop your customer base and community, streamlining processes in getting ROI on our workflow, New business opportunities through improved workflow, Developing eMarketing/E-Business Strategy, etc. I have been in the field BPO/KPO/ITES, Typesetting, and ePublishing for 16 years, handled various projects. I am a software consultant with Master's Degree. I have sound knowledge in TEX, LATEX2ε, XMLTEX, Quark, InDesign, XML, MathML, DTD, XSLT, XSL-FO, Schema, ebooks, OeB, etc.

Sievers, Martin
Im Treff 8, 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.

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 sixteen years of experience in TEX and twenty-nine years 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) live.com
Web: http://www.latexcopyeditor.net
http://www.editingscience.net
Copyediting your .tex manuscript for readability and mathematical style by a Harvard Ph.D. Experience: edited hundreds of ESL journal articles, economics and physics textbooks, scholarly monographs, LATEX manuscripts for the Physical Review; career as professional, published physicist.

Letters

Is TEX obsolete?

Jonathan Fine

The conversion, in 2010–11, of the Summa paper mill in Finland to a Google data center is a clear sign of the growing importance of electronic media and in particular web pages in human communication. In this note we ask and provide a preliminary answer to the questions: Globally, how much server CPU time is spent running TEX or one of its descendants? And for what purpose?

For Google's data centers the answer might be zero. I don't know of any Google web application that uses TEX or descendant for back-end typesetting. The closest I know of is Google charts, which provides typesetting of TEX notation mathematics. But there is strong evidence that they are not using TEX's algorithms for this.

The major site that has back-end TEX typesetting is, of course, arXiv.org. There are also some publisher sites that use a similar service for author submissions.

WordPress and PediaPress are two other major sites using TEX or similar. WordPress allows bloggers to put LATEX notation mathematics into their posts and comments. PediaPress provides a typesetting service for Wikipedia pages, which uses LATEX as the backend.

The only other significant TEX or typesetting as a web service site I know of is MathTran (developed by myself with JISC and Open University funding). This provides, as does WordPress, translation of formulae for images, but this time intended for use on third-party web sites.

The more traditional reader might say: I agree that TEX is not widely available as a web service, but what does this have to do with it being obsolete? My view is that at present TEX and its descendants are well-established only in a few paper and PDF oriented niche areas, of which mathematics and physics research is by far the most important.

If TEX does not establish itself as a ubiquitous notation and typesetting system for mathematics on web pages, and if it does not consolidate and extend its use for server-side document typesetting, then these failings may cause the system as a whole to become obsolete. This would not be due to any inherent failings, but to a failure to provide an interface that meets the needs of electronic media, particularly the web.

◇ Jonathan Fine
Milton Keynes, United Kingdom
jfine (at) pytex dot org
http://jonathanfine.wordpress.com/

TUG Institutional Members

American Mathematical Society,
Providence, *Rhode Island*

Aware Software, Inc.,
Midland Park, *New Jersey*

Banca d'Italia,
Roma, *Italy*

Center for Computing Sciences,
Bowie, *Maryland*

Certicom Corp.,
Mississauga, *Ontario*, *Canada*

CSTUG, *Praha*, *Czech Republic*

diacriTech, *Chennai*, *India*

Florida State University,
School of Computational Science
and Information Technology,
Tallahassee, *Florida*

IBM Corporation,
T J Watson Research Center,
Yorktown, *New York*

Institute for Defense Analyses,
Center for Communications
Research, *Princeton*, *New Jersey*

MacKichan Software, Inc.,
Washington/New Mexico, *USA*

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*

Springer-Verlag Heidelberg,
Heidelberg, *Germany*

Stanford University,
Computer Science Department,
Stanford, *California*

Stockholm University,
Department of Mathematics,
Stockholm, *Sweden*

University College, Cork,
Computer Centre,
Cork, *Ireland*

Université Laval,
Ste-Foy, *Québec*, *Canada*

University of Oslo,
Institute of Informatics,
Blindern, *Oslo*, *Norway*

TUG financial statements for 2010

David Walden

The financial statements for 2010 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://www.tug.org/tax-exempt`.

Revenue (income) highlights

Membership dues revenue was up from 2009 to 2010 although our membership was down (at the end of December 2010 we had approximately 1,423 paid members); all other income categories were down.

Cost of Goods Sold and Expenses highlights, and the bottom line

Payroll, office expenses, and *TUGboat* and DVD production and mailing continue to be the major expense items. Costs were down generally, in some cases significantly.

Although overall income was down almost $5,000 year-to-year, Cost of Goods Sold and Expenses was down by over three times as much resulting in a profit for the year of almost $11,000.

Often we have a prior year adjustment that takes place early in the year to compensate for something that had to be estimated at the time the books were closed at year end; in 2010 the total of such adjustments was $1,969.

Balance sheet highlights

TUG's end-of-year asset level is down a little under $3,000 from 2009 to 2010. Although we generated a profit of almost $11,000 in 2010, we also had a cash outlay of about $11,000 in 2010 for a 2009 *TUGboat* expense (see Accounts Payable). Thus, Total Checking/Savings are less than $1,000 apart from 2009 to 2010.

The Deferred Expense of over $2,000 was cash paid out in 2009 (when we committed to the conference hotel in San Francisco) for what was actually a 2010 conference expense.

The Committed Funds come to TUG specifically for designated projects: the LaTeX project, the TeX development fund, CTAN, and so forth. They 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 2010 or previous years for 2011 and beyond. Most of this liability (the 2011 portion) was converted into regular Membership Dues for 2011 in January 2011.

The payroll liabilities are for 2010 state and federal taxes due January 15, 2011.

The change in Total Equity from 2009 to 2010 is explained as follows: Unrestricted Equity as we entered 2010 was the Total Equity from 2009 (the Unrestricted Equity from 2008 minus the 2009 loss);

Total Equity at the end of 2010 is the Unrestricted Equity with which we entered 2010 plus the Net Income (profit) from 2010.

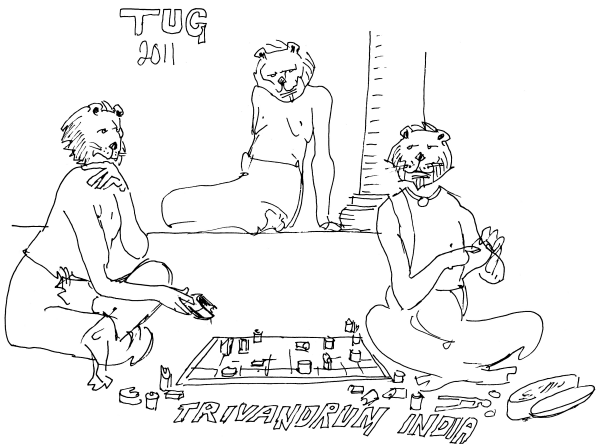
Summary

TUG remained financially solid as we entered 2010.

We did increase the fee rates (after holding them steady for three years) to cover slowly inflating expenses that we were budgeting. There was a decrease in membership, and part of that may have been because of the fee increase.

TUG continues to work closely with the other TEX user groups and ad hoc committees on many activities to benefit the TEX community.

◇ David Walden
TUG treasurer
http://tug.org/tax-exempt



TUG 12/31/2010 (vs. 2009) Balance Sheet

	Dec 31, 10	Dec 31, 09
ASSETS		
Current Assets		
Total Checking/Savings	180,613	181,596
Accounts Receivable	285	55
Other Current Assets		2,029
Total Current Assets	180,898	183,680
Fixed Assets	808	1,068
TOTAL ASSETS	181,706	184,748
LIABILITIES & EQUITY		
Liabilities		
Accounts Payable		11,000
Committed Funds	41,405	43,417
Deferred conference income		830
Prepaid member income	3,160	3,305
Payroll Liabilities	1,087	1,079
Total Current Liabilities	45,652	59,631
TOTAL LIABILITIES	45,652	59,631
Equity		
Unrestricted	125,117	128,945
Net Income	10,937	-3,828
Total Equity	136,054	125,117
TOTAL LIABILITIES & EQUITY	181,706	184,748

TUG 12/31/2010 (vs. 2009) Revenue and Expenses

	Jan - Dec 10	Jan - Dec 09
Ordinary Income/Expense		
Income		
Membership Dues	104,261	98,815
Product Sales	4,224	5,095
Contributions Income	6,515	9,253
Annual Conference	2,820	7,640
Interest Income	1,356	3,163
Advertising Income	265	315
Total Income	119,441	124,281
Cost of Goods Sold		
TUGboat Prod/Mailing	24,001	31,045
Software Production/Mailing	3,055	4,112
Postage/Delivery - Members	2,149	2,331
Conf Expense, office + overhead		1,840
JMM supplies/shipping		370
Member Renewal	523	434
Copy/Printing for members	47	234
Total COGS	29,775	40,366
Gross Profit	89,666	83,915
Expense		
Contributions made by TUG	2,000	5,000
Office Overhead	12,161	16,560
Payroll Exp	65,778	64,451
Professional Fees		230
Lucida OpenType Development	500	
Depreciation Expense	260	1,328
Total Expense	80,699	87,569
Net Ordinary Income	8,967	-3,654
Other Income/Expense		
Other Income		
Prior year adjust	1,969	-175
Total Other Income	1,969	-175
Net Income	10,936	-3,829

TUG Business

TUG 2011 election report

Nominations for TUG President and the Board of Directors in 2011 have been received and validated. Because there is a single nomination for the office of President, and because there are not more nominations for the Board of Directors than there are open seats, there will be no requirement for a ballot in this election.

For President, Steve Peter was nominated. As there were no other nominees, he is duly elected and will serve for two years.

For the Board of Directors, the following individuals were nominated: Barbara Beeton, Karl Berry, Susan DeMeritt, Michael Doob, Taco Hoekwater, Ross Moore, Cheryl Ponchin, Philip Taylor, and Boris Veytsman. As there were not more nominations than open positions, all the 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 at River Valley Technologies in India. Congratulations to all.

Board member Jon Breitenbucher has decided to step down at the end of his term. On behalf of the Board, I wish to thank him for his service, and for his continued participation through October.

Statements for all the candidates, both for President and for the Board, are appended (in alphabetical order). They are also available online at the url below, along with announcements and results of previous elections.

◇ Jim Hefferon
for the Elections Committee
http://tug.org/election

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;
- TUG committees: publications, bylaws, elections;
- chair, Technical Working Group on Extended Math Font Encoding;
- liaison from Board to Knuth Scholarship Committee 1991–1992.

Employed by 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–present.

Personal 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 since 2003 and was a board member for two terms prior to that. I am now

running for a position on the board, feeling that it was time to step down as president. However, I don't expect to materially change my efforts on behalf of TUG and TEX.

I have been on the TUG technical council for many years. I co-sponsored the creation of the TEX Development Fund in 2002, and am one of the primary system administrators and webmasters for the TUG servers. I'm also one of the production staff for the *TUGboat* journal.

On the development side, I'm currently 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 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 TEX, now freely available. I first encountered and installed TEX in 1982, as a college undergraduate.

Personal 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 essentially full time, pro bono, on TUG and TEX activities the past several years, and plan to continue doing so if re-elected as a board member.

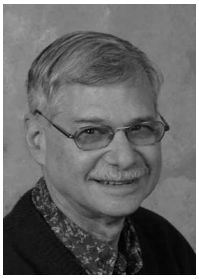
Susan Demeritt

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_ε. I continue to enjoy using LATEX 2_ε 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.

Taco Hoekwater

Biography:

Taco Hoekwater (born in 1969 in Netherlands) is the main developer of LuaTEX, and MetaPost/MPlib. He has been the first user of ConTEXt outside of PRAGMA ADE and works in tight cooperation with Hans Hagen to develop TEX engines and macros for ConTEXt ever since. Around 2005 he took over development of MetaPost, originally written by John Hobby, to implement some features needed in ConTEXt and by Polish MetaType1 font developers. During the development of ConTEXt MkIV he turned it into a library called MPlib to improve efficiency and gain speed in ConTEXt which is known for heavy use of MetaPost graphics. He has been the president of the Dutch language-oriented TEX users group (NTG) since 2009 and main editor of the user group's magazine MAPS. He was the first and main organizer of ConTEXt User Meetings, the first one being hosted in Epen, Netherlands in 2007, followed by the joint EuroTEX & 3rd ConTEXt Meeting in The Hague in 2009.

Personal statement:

After three decades, there is still a lot of life in TEX. New developments like XETEX, LuaTEX, TEXworks and also the continuously improving TEX Live help bring new life in the community. As a board member, I hope to be able to promote future extensions and applications of Knuth's amazing piece of software.

Ross Moore

Biography:

My name is Ross Moore; I am an academic mathematician, living in Sydney, Australia.

Since the mid-80s I have been a user of TEX and LATEX, mostly for mathematical applications, such as journal articles, books and proceedings volumes. The need for special content layouts has led to my involvement in the development of several packages and other software, most notably XY-pic and LATEX2HTML, both of which I have presented at TUG annual meetings.

My first TUG meeting in 1997 saw me joining the TUG Board of Directors, where I have served ever since, and wish to continue to serve for at least another term. For TUG I've worked on the Technical Council, the Publications Committee, assisted with organising annual meetings, been the contact person for the technical working groups TFAA and MacTEX (though the most important work is done by others), and administer email discussion groups (LATEX2HTML, XY-pic, XETEX). Frequently I answer queries and give advice on the 'TEX on Mac OS X' mailing list, for Macintosh users of TEX.

Currently I am working to develop TEX support for "Tagged PDF" and "Universal Accessibility" and archivability, through the PDF/A and PDF/UA formats. This is especially intricate for mathematics, which requires embedding a MathML description of the content inside the PDF document, as well as including the words for a spoken version that can be read for the benefit of the visually impaired. If TEX is to remain relevant as a publishing tool, in the context of the rapidly advancing world of communication technologies, then these technologies must be embraced and proper support developed.

Personal statement:

For the TUG board, I feel that my experience as both a TEX programmer, as well as a mathematical author and editor, provides a detailed understanding of how TEX and LATEX have been used in the past, as well as insight into new ways that the TEX family of programs will be used in coming years.

Steve Peter

Biography:

I am a linguist, publisher and designer originally from Illinois, but now living in New Jersey. I first encountered TEX as a technical writer documenting Mathematica. Now I use TEX and friends for a majority of my publishing work and work with several publishers customizing TEX-based publishing systems. I am especially interested in multilingual typography and finding a sane way to typeset all of those crazy symbolisms linguists create. As if that weren't bad enough, I also design typefaces. (Do I know lucrative markets, or what?)

I got involved in TUG via translations for *TUGboat*, where I also work on the production team. I've been on the board of directors for the past half-dozen years, and I pledge to do my best to build on the excellent work of Karl Berry as TUG president.

Personal statement:

The future of TEX and TUG lies in communication and working together to promote and sustain the amazing typographic quality associated with TEX and friends. I am especially interested in having TUG support various projects (technical and artistic) that will serve to bolster TEX and TUG's visibility in the world at large.

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.

Philip Taylor

Philip Taylor took early retirement from his post as Webmaster of one of the larger University of London colleges just over three years ago, and has since then devoted much of his time to gardening, cycling, and table-tennis. Although he retains his former professional interest in web technology, and continues to maintain and develop a number of web sites, his real passion always was, and remains, computer typesetting, for which he continues to believe that there is no better system than TEX and its offspring : ε-TEX, PdfTEX, XETEX, and LuaTEX. Having served as a member of the TUG Board of Directors for several years, he had intended to stand down at the end of his current term of office, but the recent announcement by Karl Berry that he (Karl) now intends to stand down as President has caused him (Phil) to re-consider, as he believes that the Board could be weakened if too many members were to leave at the same time.

If re-elected, Phil will continue to do his best to ensure that the needs and wishes of the ordinary TUG member are paramount when Board-level decisions need to be made.

Boris Veytsman

Biography:

I have been using TEX since 1994 and have been a TEX consultant for about six years.

Personal statement:

My goal is to make TUG a little bit more useful for the members. I think that besides our traditional benefits (*TUGboat*, software DVDs) we can do more. For example, we can convince vendors to provide exclusive discounts for our members on books, software and other products. I have done some work along these lines, and would like to continue it.

Calendar

2011

Apr 29 – May 3 — EuroBachoTEX 2011: 19th BachoTEX Conference, "Aesthetics and effectiveness of the message, cultural contexts", Bachotek, Poland. www.gust.org.pl/bachotex/bachotex2011-en

May 1 — **TUG election:** nominations due. tug.org/election

May 6 — "Graphic Design: History in the Making", St Bride Library, London, England. stbride.org/events First in a two-part series; see Nov 24-25.

May 10 – 12 — GUTenberg participation at "Solutions Linux", La Défense, Paris, France. www.solutionslinux.fr

May 31 — NTG 47th meeting, Breda, Netherlands. www.ntg.nl/bijeen/bijeen47.html

Jun 6 – Jul 29 — Rare Book School, University of Virginia, Charlottesville, Virginia. Many one-week courses on type, bookmaking, printing, and related topics. www.rarebookschool.org/schedule

Jun 19 – 22 — Digital Humanities 2011, Alliance of Digital Humanities Organizations, Stanford University, Palo Alto, California. www.digitalhumanities.org

Jul 5 – 10 — TypeCon 2011: "Surge", New Orleans, Louisiana. www.typecon.com

Jul 10 – 23 — Wells College Book Arts Center, Summer Institute, Aurora, New York. www.wells.edu/pdfs/Book_Arts_Summer_2011.pdf

Jul 14 – 17 — SHARP 2011, "The Book in Art & Science", Society for the History of Authorship, Reading & Publishing. Washington, DC. www.sharpweb.org

Jul 20 – 21 — "Towards a Digital Mathematics Library" (DML 2011), Bertorino, Italy. www.fi.muni.cz/~sojka/dml-2011.html

Aug 7 – 11 — SIGGRAPH 2011, Vancouver, Canada. www.siggraph.org/s2011

Sep 14 – 18 — Association Typographique Internationale (ATypI) annual conference, Reykjavik, Icelend. www.atypi.org

Sep 19 – 22 — ACM Symposium on Document Engineering, Mountain View, California. www.documentengineering.org

Sep 19 – 24 — The fifth ConTEXt user meeting, Porquerolles, France. meeting.contextgarden.net/2011

Sep 28 – Oct 2 — TEXperience 2011 (4th TEXperience Conference, organized by CSTUG and the Faculty of Management and Economics, Tomas Bata University in Zlín), Železnà Ruda, The Czech Republic. striz9.fame.utb.cz/texperience

Oct 14 – 15 — American Printing History Association's 36th annual conference, "Printing at the Edge", University of California, San Diego, California, www.printinghistory.org/about/calendar.php

Oct 14 – 16 — The Ninth International Conference on the Book, University of Toronto, Ontario, Canada. booksandpublishing.com/conference-2011

TUG 2011
Trivandrum, India.

Oct 19 – 21 — The 32nd annual meeting of the TEX Users Group. TEX in the eBook era. tug.org/tug2011

Sep 30 – Oct 2 — DANTE Herbstagung and 45th meeting, Garmisch-Partenkirchen, Germany. www.dante.de/events/mv45.html

Nov 24 – 25 — "Graphic Design: History in the Making", École des Beaux-Arts de Rennes, France. stbride.org/events

Status as of 1 April 2011

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 206 203-3960, e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

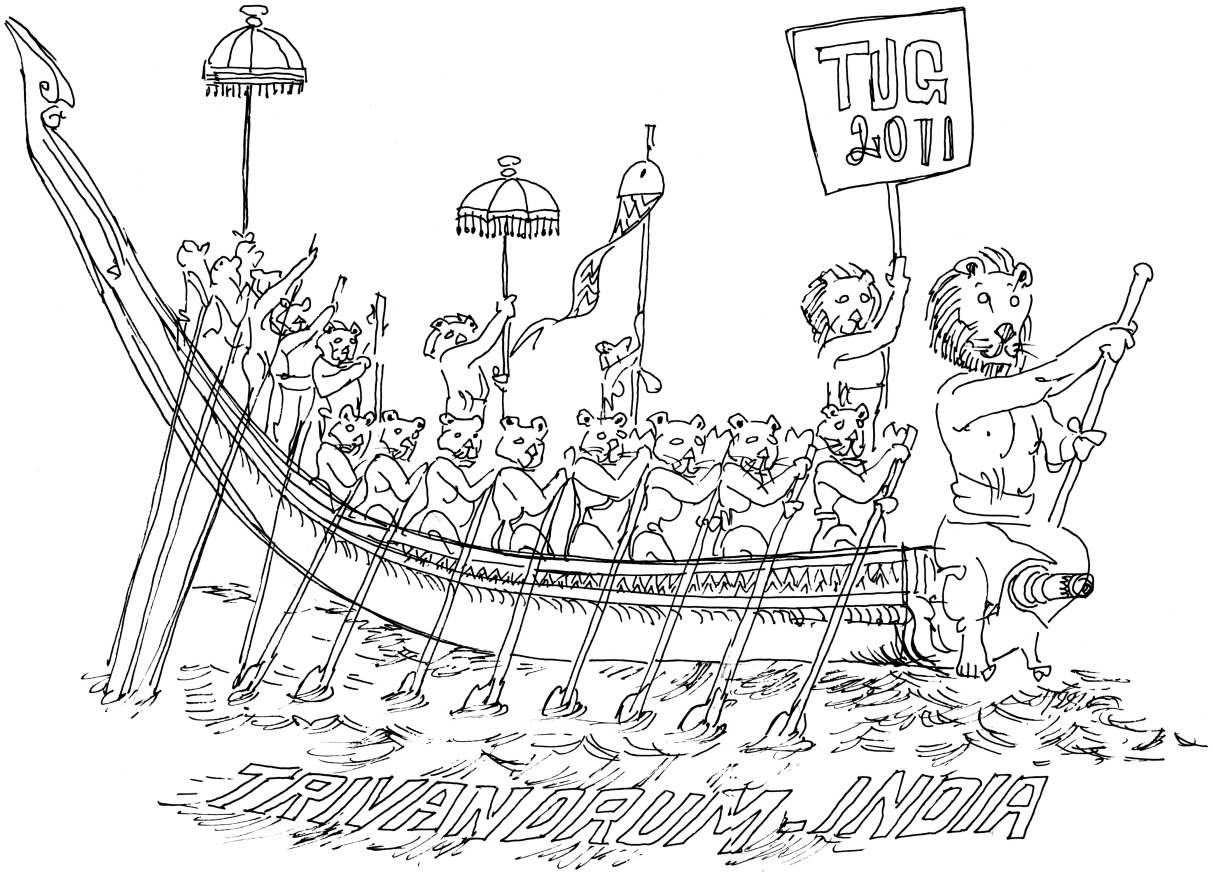
A combined calendar for all user groups is online at texcalendar.dante.de.

Other calendars of typographic interest are linked from tug.org/calendar.html.

TUG 2011: TEX in the eBook era

Presentations covering the TEX world
The 32nd Annual Meeting of the TEX Users Group

http://tug.org/tug2011 ■ tug2011@tug.org



October 19–21, 2011

River Valley Technologies
Trivandrum, Kerala
India

- July 15, 2011 — presentation proposal deadline
- August 1, 2011 — early bird registration deadline
- September 1, 2011 — preprints deadline
- October 19–21, 2011 — conference and workshop
- October 31, 2011 — deadline for final papers

Sponsored by the TEX Users Group, DANTE e.V., and River Valley Technologies.