

TUGBOAT

Volume 40, Number 1 / 2019

General Delivery	3	From the president / <i>Boris Veytsman</i>
	4	Editorial comments / <i>Barbara Beeton</i> A memorial for SPQR; Project support from UK-TUG and TUG; Installing historic T _E X Live on Unix; Converting images to L ^A T _E X: mathpix.com ; Fonts, fonts, fonts! (Helvetica Now, Study, Public Sans, Brill diacritics, Berlin typography)
	5	Noob to Ninja: The challenge of taking beginners' needs into account when teaching L ^A T _E X / <i>Sarah Lang</i> and <i>Astrid Schmölzer</i>
Tutorials	10	The DuckBoat — News from T _E X.SE: Processing text files to get L ^A T _E X tables / <i>Carla Maggi</i>
Accessibility	14	No hands — the dictation of L ^A T _E X / <i>Mike Roberts</i>
	17	Nemeth braille math and L ^A T _E X source as braille / <i>Susan Jolly</i>
Software & Tools	22	Both T _E X and DVI viewers inside the web browser / <i>Jim Fowler</i>
	25	Markdown 2.7.0: Towards lightweight markup in T _E X / <i>Vít Novotný</i>
	28	New front ends for T _E X Live / <i>Siep Kroonenberg</i>
	30	TinyTeX: A lightweight, cross-platform, and easy-to-maintain L ^A T _E X distribution based on T _E X Live / <i>Yihui Xie</i>
	33	Extending primitive coverage across engines / <i>Joseph Wright</i>
	34	ConT _E Xt LMTX / <i>Hans Hagen</i>
	38	Bringing world scripts to LuaT _E X: The HarfBuzz experiment / <i>Khaled Hosny</i>
L^AT_EX	44	L ^A T _E X news, issue 29, December 2018 / <i>L^AT_EX Project Team</i>
	47	Glossaries with <code>bib2gls</code> / <i>Nicola Talbot</i>
	61	T _E X.StackExchange cherry picking, part 2: Templating / <i>Enrico Gregorio</i>
	69	Real number calculations in L ^A T _E X: Packages / <i>Joseph Wright</i>
Macros	71	Real number calculations in T _E X: Implementations and performance / <i>Joseph Wright</i>
Electronic Documents	76	T _E X4ht: L ^A T _E X to Web publishing / <i>Michal Hoftich</i>
	82	TUGboat online, reimplemented / <i>Karl Berry</i>
Book Reviews	85	Book review: <i>Never use Futura</i> by Douglas Thomas / <i>Boris Veytsman</i>
Abstracts	88	<i>Die T_EXnische Komödie</i> : Contents of issue 1/2019
	88	<i>Zpravodaj</i> : Contents of issue 2018/1–4
Hints & Tricks	89	The treasure chest / <i>Karl Berry</i>
Cartoon	91	Comic: Punctuation headlines / <i>John Atkinson</i>
Advertisements	91	T _E X consulting and production services
TUG Business	2	TUGboat editorial information
	2	TUG institutional members
	93	TUG financial statements for 2018 / <i>Karl Berry</i>
	94	TUG 2019 election
News	96	Calendar

T_EX Users Group

TUGboat (ISSN 0896-3207) is published by the T_EX Users Group. Web: tug.org/TUGboat.

Individual memberships

2019 dues for individual members are as follows:

- Trial rate for new members: \$20.
- Regular members: \$105.
- Special rate: \$75.

The special rate is available to students, seniors, and citizens of countries with modest economies, as detailed on our web site. Members may also choose to receive *TUGboat* and other benefits electronically, at a discount. All membership options described at tug.org/join.html.

Membership in the T_EX 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 carries with it such rights and responsibilities as voting in TUG elections. All the details are on the TUG web site.

Journal subscriptions

TUGboat subscriptions (non-voting) are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate for 2019 is \$110.

Institutional memberships

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

Trademarks

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

[printing date: May 2019]

Printed in U.S.A.

Board of Directors

Donald Knuth, *Grand Wizard of T_EX-arcana*[†]

Boris Veytsman, *President**

Arthur Reutenauer*, *Vice President*

Karl Berry*, *Treasurer*

Susan DeMeritt*, *Secretary*

Barbara Beeton

Johannes Braams

Kaja Christiansen

Taco Hoekwater

Klaus H \ddot{o} ppner

Frank Mittelbach

Ross Moore

Cheryl Ponchin

Norbert Preining

Will Robertson

Herbert Vo β

Raymond Goucher, *Founding Executive Director*[†]

Hermann Zapf (1918–2015), *Wizard of Fonts*

** member of executive committee*

† honorary

See tug.org/board.html for a roster of all past and present board members, and other official positions.

Addresses

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

Telephone

+1 503 223-9994

Fax

+1 815 301-3568

Web

tug.org
tug.org/TUGboat

Electronic Mail

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

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

Technical support for
T_EX users:
support@tug.org

Contact the
Board of Directors:
board@tug.org

Copyright © 2019 T_EX Users Group.

Copyright to individual articles within this publication remains with their authors, so the articles may not be reproduced, distributed or translated without the authors' permission.

For the editorial and other material not ascribed to a particular author, permission is granted to make and distribute verbatim copies without royalty, in any medium, provided the copyright notice and this permission notice are preserved.

Permission is also granted to make, copy and distribute translations of such editorial material into another language, except that the T_EX Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.

Kedit is the only text editor I have ever used. [...] Kedit did not paginate, italicize, approve of spelling, or screw around with headers, wysiwygs, thesauruses, dictionaries, footnotes, or Sanskrit fonts. [...] Kedit was “very much a thing of its time,” and its time is not today. I guess I’m living evidence of that.

John McPhee
Draft No. 4: On the Writing Process
(2017)

TUGBOAT

COMMUNICATIONS OF THE T_EX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 40, NUMBER 1, 2019
PORTLAND, OREGON, U.S.A.

TUGboat editorial information

This regular issue (Vol. 40, No. 1) is the first issue of the 2019 volume year. The second issue this year will be the TUG'19 conference proceedings; the submission deadline is August 18. The deadline for the third issue is September 30.

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 (tug.org/store), and online at the *TUGboat* web site (tug.org/TUGboat). Online publication to non-members is delayed for one issue, 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 assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

TUGboat editorial board

Barbara Beeton, *Editor-in-Chief*
 Karl Berry, *Production Manager*
 Robin Laakso, *Office Manager*
 Boris Veytsman, *Associate Editor, Book Reviews*

Production team

William Adams, Barbara Beeton, Karl Berry,
 Kaja Christiansen, Robin Fairbairns, Clarissa Littler,
 Steve Peter, Michael Sofka, Christina Thiele

TUG Institutional Members

TUG institutional members receive a discount on multiple memberships, site-wide electronic access, and other benefits:

tug.org/instmem.html

Thanks to all for their support!

American Mathematical Society,
Providence, Rhode Island

Association for Computing
 Machinery, *New York, New York*

Aware Software, *Newark, Delaware*

Center for Computing Sciences,
Bowie, Maryland

CSTUG, *Praha, Czech Republic*

Harris Space and Intelligence
 Systems, *Melbourne, Florida*

Institute for Defense Analyses,
 Center for Communications
 Research, *Princeton, New Jersey*

Maluhy & Co., *São Paulo, Brazil*

Marquette University,
Milwaukee, Wisconsin

Masaryk University,
 Faculty of Informatics,
Brno, Czech Republic

MOSEK ApS,
Copenhagen, Denmark

Nagwa Limited, *Windsor, UK*

New York University,
 Academic Computing Facility,
New York, New York

Overleaf, *London, UK*

StackExchange,
New York City, New York

Stockholm University,
 Department of Mathematics,
Stockholm, Sweden

T_EXFolio, *Trivandrum, India*

TNQ, *Chennai, India*

University College Cork,
 Computer Centre,
Cork, Ireland

Université Laval,
Ste-Foy, Québec, Canada

University of Ontario,
 Institute of Technology,
Oshawa, Ontario, Canada

University of Oslo,
 Institute of Informatics,
Blindern, Oslo, Norway

V_TE_X UAB, *Vilnius, Lithuania*

TUGboat advertising

For advertising rates and information, including consultant listings, contact the TUG office, or see:

tug.org/TUGboat/advertising.html

tug.org/consultants.html

Submitting items for publication

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

The *TUGboat* style files, for use with plain T_EX and L^AT_EX, are available from CTAN and the *TUGboat* web site, and are included in common T_EX distributions. We also accept submissions using ConT_EXt. Deadlines, templates, tips for authors, and more, is available at tug.org/TUGboat.

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 suitable 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 T_EX community in general.

If you have such items or know of any that you would like considered for publication, please contact the Publications Committee at tug-pub@tug.org.

From the president

Boris Veytsman

In the beginning of this century the US Federal Aviation Administration decided to overhaul elements of the air traffic safety infrastructure, introducing mandatory Automatic Dependent Surveillance Broadcast (ADS-B).¹ The company I worked for at that time was chosen to implement this system. I remember the words by the program leader Glen Dyer at that time,

You know, I always envied the civil engineers. Just imagine walking with your grand-kids, stopping by a beautiful bridge and telling them, “This bridge was designed by me.” Now we have our chance to design our “bridge”. We are doing something that will endure.

He was right: ADS-B is being used in all flights over the US, and is going to stay. It is planned that this technology will still be employed in the foreseeable future, so we definitely will be able to talk about it with our grandchildren.

I recalled this episode recently while reading a tweet from Nathan Lane (@straightedge):²

Redoing my slides in LaTeX because I fear my MA students won't see me as a true scholar unless I demonstrate mastery of a 1980s typesetting markup language

This tweet can be interpreted in different ways. One may consider it a complaint about the strange allure of obsolete solutions — shouldn't we use something newer and shinier than an invention of the nineteen eighties? Do the grandchildren of the first users of \TeX still need to learn it?

This complaint was transmitted to thousands of people on Twitter. Twitter itself started in 2006, so it is relatively new. However, it relies on the HTTP(S) protocol based on the work started by Sir Tim Berners-Lee at CERN in 1989. Moreover, HTTP messages are distributed by the TCP/IP network protocol, developed in 1974. Thus a more accurate version of the tweet might sound like this:

Using protocols developed in 1970s and 1980s to complain about the requirement to show the mastery of a 1980s typesetting markup language for being considered a true scholar.

I would like to make an observation based on this (and many other) episodes. The pace of change for a technology strongly depends on how “deep” the technology is. A user interface for Twitter changes

fast; the underlying software layers are much more stable, and the lower we go into them, the slower they change. In the same way the safety features of aviation are much more stable than aircraft cabin interiors. Thus the relatively slow pace of \TeX 's evolution might be a result of its place in the computer infrastructure: it provides the foundation for exchange of information.

This, of course, does not mean that \TeX is not going to change: it *is* changing now and will continue to change in the future. To continue the analogy with network protocols, neither HTTP nor IP stayed still in recent years. Among the changes were the addition of security layers, transition to IPv6 with a huge number of new addresses, extensions for space-based communications, etc. This is definitely not your parents' Internet Protocol any more!

Similarly \TeX today is definitely not the 7-bit \TeX of 1980s. The new engines with native Unicode support transformed the way we deal with multiple languages. They provide new primitives allowing new possibilities for \TeX programmers. New(er) macro systems like the (continuously developed) Con \TeX t and L \TeX 3 give much needed flexibility to document designers and package writers. Graphics packages like PSTricks and TikZ provide ways to flexibly incorporate non-textual information into \TeX documents. The “bridge” from the beginning of this essay is being constantly overhauled and improved.

A living architectural or engineering structure like a historic bridge or building provides an interesting challenge to the community. Its beauty and design must be preserved, but it must also provide for the changed needs of the users. Thus it must be updated — but carefully and deliberately.

Similarly we at TUG — and the general \TeX users community — have the dual duty to preserve the integrity of \TeX — and to steer its development to the ever changing needs of the typesetting community. This issue of *TUGboat* can be viewed as a report of our continuing efforts to perform this duty. The upcoming 2019 edition of the \TeX Collection, with \TeX Live, Mac \TeX , MiK \TeX , and CTAN, is another artifact of these efforts.

I feel that all of us: developers, programmers, users, are participating in the maintaining and improvement of a beautiful important edifice slated to endure. \TeX , created by Don Knuth, is now a community-based project. Let us make it shine!

Happy \TeX ing!

◇ Boris Veytsman
borisv (at) lk dot net
<http://borisv.lk.net>

¹ www.faa.gov/nextgen/programs/adsb
² twitter.com/straightedge/status/1118834149165428736

Editorial comments

Barbara Beeton

After long employment at the American Math Society, I retired on February 5. My term on the TUG board has been extended for four more years, and I expect to remain editor of *TUGboat*, as long as I can continue to do a creditable job. It's been an interesting run.

A memorial for SPQR

Sebastian Rahtz had a favorite place — The Protestant Cemetery in Rome (Il Cimitero Acattolico di Roma). A bench with a tribute to Sebastian has now been installed in the cemetery, providing a place for visitors to rest and contemplate.



The director of the cemetery sent this report:

[...] We put it in the Parte Antica some weeks ago and people immediately started sitting on it! I had to ask them to move so I could take the photo! It is a lovely addition to the Cemetery. We put it near the Garden Room, and people are allowed to move it around as they like. In winter they put the benches in the sun, in summer they like to sit under the shade of the trees. The benches are a wonderful way for people to relax (we all do far too little of that these days) and a possibility for people to chat and renew or make new friendships. I like to think that Sebastian would have loved that.

The Protestant Cemetery is the final resting place of many visitors to Rome who died there; it may be best known as the gravesite of the English poets Keats and Shelley. It is one of the oldest burial grounds in continuous use in Europe, and in 2016, its 300th anniversary was celebrated. The website for the cemetery, www.cemeteryrome.it, provides a virtual visit, or information to plan for an actual visit.

Project support from UK-TUG and TUG

The UK-TUG committee has posted a reminder that there is a standing call for applications for project funding. Details can be found at uk.tug.org/about/funding/, and proposals sent to uktug-committee@uk.tug.org.

TUG also offers project funding: see tug.org/tc/devfund.

Installing historic T_EX Live on Unix

For Unix only, Péter Szabó has written scripts for convenient installation of historic T_EX Live distributions. The years now covered are 2008 through 2018. The scripts and information are at github.com/pts/historic-texlive. This can be a welcome backup in case you have an older document that won't run with the latest distribution.

All T_EX Live releases (and much more) are available at ftp.tug.org/historic/systems/texlive and ftp.math.utah.edu/pub/tex/historic. (Additional mirrors would be most welcome.) Péter's new scripts download from this archive.

Converting images to L^AT_EX: mathpix.com

The `mathpix` application purportedly allows one to take a screenshot of math and paste the L^AT_EX equivalent into one's editor, "all with a single keyboard shortcut". This tool (mathpix.com) appears to hold more promise than other attempts. We have solicited a review, which we hope will appear later this year.

Fonts, fonts, fonts!

More than the usual number of font-related announcements have appeared since our last issue. Many notices came via CTAN, but those are ignored here; instead we briefly mention several gleaned from other sources. It isn't known whether (L^A)T_EX support is available for any of them, but it's likely to happen sooner or later.

We finish up with a more expansive review of a website devoted to "fonts in the wild" — photos of lettering found on surfaces in a city environment.

Helvetica redesign! On 9 April 2019, Monotype introduced the Helvetica Now family, a redesign of the venerable and ubiquitous typeface. The last redesign resulted in Neue Helvetica 35 years ago. The announcement describes the changes as expressly tuned "for the modern era". The new rendition includes three optical sizes and "a host of useful alternates". Read the full text and watch the video at tug.org/1/helvnow.

Making Study: New clothing for the twenty-six leaden soldiers. Study is a new typeface, "completed" by Jesse Ragan based on a design by Rudolph Ruzicka, a Czech type designer active in the 1940s–60s. Ruzicka's typefaces, produced in metal by Mergenthaler Linotype and used for books in their heyday, were never effectively reworked for newer technologies, and are not much in use today. However, although he completed only two designs, Ruzicka never stopped coming up with new ideas. A collection of these ideas was published

in 1968, when Ruzicka was 85, in a work entitled *Studies in Type Design*. One of the designs, Study, was clearly well developed, but lacking kerns and normalization of features such as stem and hair-line thickness, which are necessary for the typeface to be usable. It is this design that Ragan chose to complete. The story is fascinating. Read it at xyztpe.com/news/posts/design-notes-study.

Public Sans — The U.S. government gets involved. As its name implies, this is a sans serif typeface intended for public use. It is part of “a design system for the federal government”, to “make it easier to build accessible, mobile-friendly government websites for the American public.” Nine weights are available, and there is an invitation to contribute to its development on GitHub. Details, such as they are, are at public-sans.digital.gov and links therein.

Diacritics to die for: Brill. The Brill typeface, designed and implemented principally by John Hudson of Tiro Typeworks, is a custom design for the Dutch scholarly publisher in Leiden now known as Brill. This firm has a history of more than 330 years of publishing in many languages and scripts, including the International Phonetic Alphabet (IPA). As if the different scripts weren’t enough of a problem, the proliferation of diacritics — often multiply applied to the same base character — exceeded anything that I had ever seen. This came to my attention in a question on tex.sx asking for help in reproducing the image on page 22 of these slides: www.tiro.com/John/Hudson-Brill-DECK.pdf.

The implementation of diacritic placement uses the GPOS table of OpenType fonts. This can be accessed with $X_{\text{g}}\text{T}_{\text{E}}\text{X}$, with some limitations, as pointed out by Khaled Hosny in his response to the referenced question (tex.stackexchange.com/q/485523). If you think math is complicated, read through the entire slide presentation and marvel.

Brill has made their eponymous types available for non-commercial use by individuals.

Berlin Typography. The website “Berlin Typography” (see berlintypography.wordpress.com) is a wide-ranging photo essay showing off signs and other examples of text that appear on buildings and other surfaces in Berlin. The series was started in 2017 and is still active in 2019. Each post highlights a different topic; the most recent posting examines “Shoes and their makers: Footwear in Berlin”. Other selections include street signs, shops offering various merchandise and services, plaques, text on stones and grates in the pavement, and much, much more.

Perhaps my favorite collection is [.../2017/03/06/berlins-bridge-typography](http://2017/03/06/berlins-bridge-typography) — typography on Berlin’s bridges. Search for the term “blackletter” to see the photo that would have been included here, had our attempt to obtain permission been successful. (We were unable to find contact information.)

In 2018, an interview with the creator of the blog, Jesse Simon, appeared online: “Celebrating Berlin’s Typography, before it vanishes: A tour of the city’s most striking signs”, by Anika Burgess. It’s well worth a look: www.atlasobscura.com/articles/berlin-signs-typography.

These essays inspire one to be aware of one’s surroundings — look both up and down — and notice what’s there in plain sight.

◇ Barbara Beeton
<https://tug.org/TUGboat>
 tugboat (at) tug dot org

Noob to Ninja: The challenge of taking beginners’ needs into account when teaching \LaTeX

Sarah Lang and Astrid Schmölder

On the 26th of February this year, the first guest post “Confessions of a \LaTeX Noob”¹ was posted on the \LaTeX Ninja blog (<https://latex-ninja.com>). Quite unexpectedly, it seems that the article hit a nerve in the $(\text{\LaTeX})\text{T}_{\text{E}}\text{X}$ community since we were showered with feedback and encouragement in the days that followed. Many long-standing experts told us, to our great surprise, that they were very interested in hearing the experiences of a less advanced user wanting to learn. Also, we were greeted by a community of $(\text{\LaTeX})\text{T}_{\text{E}}\text{X}$ enthusiasts who felt genuinely sorry for the problems the \LaTeX Noob had experienced and the feelings of not belonging which they had expressed. This is how it came that we were invited to contribute a similar essay to *TUGboat*. Since the roles of the \LaTeX Ninja and the \LaTeX Noob have turned out to work well to make our point, we decided to keep them.²

So today, we wanted to talk about how to take beginner’s needs into account when teaching \LaTeX or other “outreach” activities such as answering questions on StackOverflow, etc.

¹ latex-ninja.com/2019/02/26/guest-post-confessions-of-a-latex-noob/

² For the purposes of getting in touch with us, we can be reached as our real life selves, Sarah Lang and Astrid Schmölder. We are happy to be able to work as early career scholars at the University of Graz (Austria) in the field of the Humanities. For questions and comments directly regarding \LaTeX , please contact us via [the.latex.ninja \(at\) gmail.com](mailto:the.latex.ninja@gmail.com).

Advanced users hardly remember how they felt starting out

While this would be quite the challenge in and of its own, we encounter the first problem, which is that the needs of beginners become increasingly obscure to us the more advanced we ourselves become. Here, the only option is to enter in an active dialogue with those who are still at the beginning of their journey and listen very carefully and attentively for how they want to be helped and what they need help with.

Making learning failsafe by minimizing initial hurdles

Existing sources like `lshort` and the great books on learning \LaTeX might already be too voluminous for a beginning user. Not everybody likes to read long documents. I love to read in general but the length of `lshort` would definitely be enough to deter me from trying to learn \LaTeX , if I had to start from scratch again.

When I first started, my then-partner handed me their own template and helped with the start-up work of converting their technological citation style to a Humanities citation style. Then I used \LaTeX for quite a while before ever reading up on it again. This practical introduction without a lot of background reading worked very well for me.

Tools like Overleaf can be part of this, since they allow the interested newbie to explore \LaTeX without the hassle of having to install \LaTeX under Windows. This point can't be stressed enough! New \LaTeX users should only try a scary thing like installing \LaTeX under Windows once they are already hooked. We don't want to lose them before they get started.

Problem: Existing sources are not geared towards the needs of beginning learners. Resources tend not to be very didactical. Many reference works exist, but few resources are fit for a beginner. Or don't lead the beginner very far. Or not accessible unless you are willing to do a lot of introductory reading.

Problem: Many tutorials exist, but most resources are a) lengthy, b) too detailed (partly the reason why they're too long) or c) only cover basic concepts. Information on how to learn advanced skills, or even what advanced skills in \LaTeX might be, is hard to find even after thorough searches.

Leveraging the principle of the “minimum effective dose”

When I first taught some \LaTeX basics to university students in an introductory Bachelor's class on annotation (mostly XML) in the Digital Humanities, I was surprised to find out that these new users were not at all afraid of \LaTeX , as one is often led to believe.

I think, the “fear” or excessive respect regarding \LaTeX is not due to the nature of \LaTeX itself. So it must be some part of the community generating this reluctance.

From my own everyday life, I gather that it is often those who are not users of \LaTeX themselves and feel bad about this alleged lack of knowledge on their part who tend to talk badly about \LaTeX . Here, the standard arguments are that \LaTeX is overly complex, difficult, a hassle, and nobody really needs it anyway. Since these people seem to be rather widespread, the (\LaTeX) community should be more active in showing that the opposite is true. This can be done by providing easy introductory tutorials (like “Learn \LaTeX in three minutes”³) which stress that \LaTeX need not be complicated — since this is an impression many people get from extensive reference-like tutorials.

Often, people just want to know “How can I get started right now?” and not after three hours of introductory reading plus supplemental Internet research. They don't care that they were not taught all the details. They are grateful for a concise introduction. So the question of “What does one need to know to start using \LaTeX ” is not “What will an average user need to know in total” but rather “What is the (absolute) minimum amount of knowledge I need to have to get started”.

Tim Ferriss calls this the “minimum effective dose”. This is so effective and fail-safe because it requires minimal effort while at the same time limiting choices — and thus common sources of failure — to a minimum. As he explains in *The 4-Hour Chef*, he suggests employing two principles to successfully leverage the minimum effective dose (MED), “failure points” and a “margin of safety”. In his case, “failure points” are a collection of reasons why people give up on cooking. Most of them revolve around too many things you need to have, know or do, as well as overall excess complexity. Assessing the “margin of safety” means “How badly can you do something and still get incredible results (relative to ‘normal standards’)?”

In the case of learning \LaTeX this translates to: How few elements do I need to know to reap some benefits of transitioning to \LaTeX ? How can I reduce complexity to prevent chaos resulting from lack of experience? I usually recommend not installing \LaTeX

³ latex-ninja.com/2018/12/11/

jumpstarting-learn-latex-in-3-minutes/

Not to say that the Ninja blogs are didactically perfect, they just serve as examples and starting points for the endeavour of creating a curriculum to learn \LaTeX , especially for people who don't have a background in technology.

on your own machine yet, to minimize the initial hurdles. When just starting out, tools like Overleaf will suffice. You don't need to have everything installed on your computer at first. Another problem is that common reference-style tutorials seldom respect didactical principles like didactical reduction, the art of hiding unnecessary detail. Hiding details is generally not something computer people approve of. But it is absolutely necessary for good teaching to hide (currently) irrelevant detail from the learner until they are ready.

Challenges and remedies

Problem: Getting help can be tricky; you don't want to look like an idiot and, especially as someone from a non-technical background, you constantly have to defend your choice to use \LaTeX (to users and non-users alike). “Why would a Humanities person want to use \LaTeX anyway? You don't need it and you're not up for it” are the most common insults a Humanities person might have to endure after choosing \LaTeX .

My friend who agreed to play the part of the \LaTeX Noob is an archaeologist and can speak from experience. She writes her thesis in \LaTeX since archaeologists, while still largely narrative-based Humanists, need a lot of image evidence, and this can result in a notorious set of problems typically encountered when creating 300-page-long, image-rich documents in MS Word. So she has as much of a valid reason for wanting to use \LaTeX as any mathematician. After all, \LaTeX has more to offer than math mode.

She is not a new user either, but she still feels like a Noob. And not in a good way but with the connotation of being an incompetent idiot and the fear she might never escape this unpleasant state of limbo. This is, however, not due to the fact of her actually being a hopeless idiot — she is quite adept with a lot of technical and even some programming matters — but much rather due to the way newbies tend to get treated on the Internet.⁴

Problem: Noob doesn't speak “nerd talk”. Tutors need to understand the people they teach. Mediators are needed between technical and non-technical people (kind of like Digital Humanists). Tech talk is not understandable to a newbie — this extends to the content as well as the way of speaking.

Problem: The constant demand to justify your choice to use \LaTeX can stop newbies from asking

for help. And if they don't succeed with their own experiments, they quit their experiments. It's only a matter of time until they quit \LaTeX completely. Often, the “Why would you want to use \LaTeX anyway?”, coming from users and non-users alike, bears the tone of accusation and aggression. Often, it is the answer new users receive when they ask a question, instead of the answer to the actual question they had asked.

Problem: Confusion generated by “insider knowledge” which can be quite hard to come by (like understanding what a float does). Noobs aren't even aware there might be problems with that, and thus don't know where to find help or look when trying to locate a bug.

Problem: After they have overcome their difficulties and proudly share their code comes the next letdown, from being ridiculed for non-perfect, but still working, code which took hours to write. Not helpful. Good style is like good fashion sense: It's nice to have but a luxury not everybody can afford.

Problem: People promise to follow up on a question by sending a code example but they never do. No follow-ups, no real intention to help? These promises might leave a newbie waiting for an email which never comes. They feel: “If you just want to show me how good you are, fine. But I really wanted your help, otherwise, I would not have asked.”

Problem: Then there is the issue of having to keep your debugging sessions and \LaTeX problems secret from non- \LaTeX -using colleagues. “Why waste your time?”, they ask reproachfully. “Shouldn't you be focusing on finishing your PhD thesis?” And maybe, one day, the Noob will come to the conclusion that yes, they need to focus on their PhD. Perhaps \LaTeX just wasn't for them.

An atmosphere needs to be created where new and progressing users are actively included in the community rather than seeing them as noobs who are not “real users”. Nobody has to prove themselves in order to be a “valid member” of the (\LaTeX) community. They just need to be passionate about (\LaTeX) and if this is the case, a learning process will ensue anyway.

Problem: Often, advanced users trying to help newbies don't even see what the actual problems are. Those could be confusion about which editor to choose, what in the world a “minimal working example” or “Lorem Ipsum” might mean or how to correctly ask a question on StackOverflow. Or things that might be deemed self-evident, something way more basic than expected. If the Noobs are lucky, they find the explanations on the Internet.

⁴ We summed up and expanded the main arguments of the original blog post in the following text, trying to isolate problems and offer solutions. For the personal account of her experiences, please consult the original blog post.

A glossary of these basics might be of help, like a “ \TeX nical literacy 101” worksheet. It might also help raise awareness as to what the “blank slate” consists of that a new user starts out with. Not all Noobs come from the field of technology where a basic knowledge of “digital literacy” is the norm. And it is not their fault that they don’t. The definition of a Digital Humanist basically is someone with knowledge of the technical and Humanities’ worlds simultaneously who can act as an interpreter. A technician might need a mediator in order to communicate effectively with completely non-technical folk.

Being tutored is a lot like women in relationships. Often, you just want to be heard and feel seen. Getting a ready-made solution is not what’s wanted. It’s more about being acknowledged with and validated in your needs and problems. For \LaTeX Noobs this might translate to feeling that your struggle is okay and valid and being reassured that you can do it and that people are there to help. A newbie does not expect you to do the work for them (at least most newbies don’t), but rather that you will enable them to find their own way. And that you are okay with whatever way works for them, even though this might not be the way you would choose yourself.

Let’s help Noobs to help themselves rather than lecture them on what’s good for them without explaining why. They want to know the facts so they can decide for themselves.

Ways to rise to the challenge — A manifesto

1. Be nice to new users.

Don’t tolerate other people’s bad behaviour towards Noobs. Noob should not be a curse word anymore. This is also why we kept the name for the persona in the blog. Noob just means newbie and we don’t see anything wrong with that. If anything, the (\LaTeX) community is enriched by every new user, no matter how little knowledge they might have. To the contrary, we invite all to step forward and ask for help. Starting to learn a new skill is a great new opportunity and we would like to see many more new users becoming confident in their \LaTeX skills. Let’s actively discourage fellow commentators when they are being rude. They might not realize this since Internet communities often encourage rough behaviour among “nerds”. But your silence will make the newbie feel like everyone thinks that way. Speaking up is all that’s needed to at least minimize bad experiences like the ones from the “Confessions of a \LaTeX Noob” post.

2. Create tutorials according to the real-life use case of the new user.

Rather than offering tons of (probably irrelevant) information in a reference to the newbie who is not yet at the stage where they can make effective use of references, offer tutorials which explain something they might want to do. Like “minimal skills to write a Humanities’ seminar paper” which can be learned in 10 minutes. The newbie might be ready to create their first document and do some basic things. But they most certainly are not yet at the point where they can write up a whole workflow according to their needs. This is fine if they have no special needs. But most users probably actively turn away from WYSIWYG editors like Word precisely because they have a special need they hope transitioning to \LaTeX will meet.

Existing tutorials often show generic snippets of how certain effects can be achieved. But it might be too much to ask of a new user to look everything up on their own and assemble it together in one document. Sadly, many common use cases, like creating (personalized) transformations from Word to \LaTeX , are not represented in tutorials so much. These are the skills I talk about when I ask for more teaching on Advanced \LaTeX . It might not seem advanced technically, but it is advanced in the way that you need to stitch a whole workflow together, plan your project and need to know where to find certain bits of information, etc. The first step from beginner to intermediate/advanced user is learning how to independently realize a project like this and create a document which matches your needs in a specialized use case.

3. Don’t ask them to plunge into cold water and completely move away from their previous WYSIWYG editors.

Don’t stop helping them only because they wish to also still use MS Word. They might only want to use \LaTeX sparingly and selectively. Even if you don’t understand this, try to understand them. Help them achieve what their old tool could do in \LaTeX if you feel the need to convince them. Often, seasoned \LaTeX users who don’t take part in more common ways of typesetting anymore (\LaTeX is, after all, still a specialist thing), don’t see why a transitioning user might want their \LaTeX document to look like the Word output they were used to. But be tolerant: This might also be due to pressure of their own community. In the Humanities, handing in documents in .docx formats is practically unavoidable. Try to be understanding or else the newbie in transition might just go back to their old ways.

4. Help new users in a way they want to be helped, not in the way you think they should do things.

This is probably the most crucial point people tend to misunderstand about empowerment. Trying to “forcefully lift people up from a seemingly superior way of doing things” is not empowerment.⁵ Helping someone make their own choices by showing them all the options available is! Do your utmost to try to avoid coming across as arrogant and exclusive. Formulate explanations in a way to encourage questions rather than leaving people humiliated about their lack of knowing better.

When spelled out like this, it sounds a little insulting, but I feel that we often don’t realize we act in those ways. Becoming aware of this will probably be enough to change the behaviour for good. It is only too easy to make a newbie feel stupid or to sound condescending over the Internet. And nobody’s perfect. But it is well worth trying to make an effort. Help given should be empowering, mentor-like and not patronizing. You are talking to adults, after all (or at least, mostly). They can make their own choices.

5. Long-term: Build curriculum so new users can acquire “the L^AT_EX skill” more conveniently.

Decide criteria of what one should know before offering L^AT_EX-based services. Show new users how using L^AT_EX will not only prettify their documents (which, we agree, is very important, of course) but also how mastery of L^AT_EX can be an asset in one’s CV and which kinds of services one could offer. If the community was able to offer a more systematic way of “initiation” to the community, this might encourage new users to start L^AT_EX.

Conclusion: Not all Noobs are made equal

If you’re doing technology or maths, nobody will question your choice to use L^AT_EX. If you don’t, everybody will. Not all Noobs are made equal. A Noob from technology already comes with the ability to understand “tech-talk”, “nerd speech”, basic workings of StackOverflow and the like. A Noob from the humanities does not. People from these two different fields will have completely different needs.

⁵ Of course, this is formulated in a polemical way and I don’t assume that you, dear reader, are this way. I merely wanted to stress it a little more than necessary to get the importance of the point across. Too often, we get caught up in our own opinions and forget that other people might think differently and that it is their good right to do so.

Resources are needed to serve as mediators, facilitating the early stages of learning for those without a technical background. Extra care for Noobs without a technical background is all the more pressing for the L^AT_EX community because they will probably already get discouraged from using L^AT_EX by basically everyone else around them. It is therefore crucial that teachers, helpers and teaching materials do their very best to encourage them. We have (pro-)actively to make up for all the unsolicited discouragement they are already receiving if we don’t want to lose these new users.⁶

Offering detailed tutorials for subjects typically needed by a non-technical newbie, also explaining seemingly ‘obvious’ bits of information someone without a technical background might not have heard about — and coming up with a set of good reasons why L^AT_EX is a great tool for non-technical people as well — would be a step in the right direction. These reasons, however, should not be the same ones generally aimed to convince tech-savvy newbies (“formula support”). If we reuse the same arguments, it is likely newbies will feel that they don’t apply to them.

Of course, the general reason of “superior typesetting” is a main argument for L^AT_EX. But it is also one likely to only be convincing to those who are already convinced of the importance of good typesetting. So this argument, while good at heart, will not convince anyone who is not already convinced. After all, learning to handle L^AT_EX is a lot of work for a newbie, so the reasons should be a lot more pragmatic. If we can show Humanities people how they can do their everyday tasks better using L^AT_EX, this would be a great improvement. Also, the more visible the L^AT_EX community becomes in the non-technical world, the less non-technical newbies will have to justify themselves for wanting to use L^AT_EX. Therefore, promote L^AT_EX!

◇ Sarah Lang and Astrid Schmörlzer
 the.latex.ninja (at) gmail dot com
<https://latex-ninja.com>

⁶ Editor’s note: The TUGboat editorial wish list, tug.org/TUGboat/wish.html, has long had the item: *More tutorials and expository material, particularly for new users and users who aren’t intending to become T_EX wizards.* Submissions welcome!

The DuckBoat — News from T_EX.SE: Processing text files to get L^AT_EX tables

Herr Professor Paulinho van Duck

Abstract

In the first part of this installment, Prof. van Duck will talk about the distribution of upvotes by topics on T_EX.SE. In the following Quack Guide, you will find out how to easily process text files to create professional L^AT_EX tables.

1 Quack chat

Hi, (L^A)T_EX friends!

An amazing event happened on October 20th, 2018, in Rome, Italy: my first talk! To tell the truth, since I was too shy to speak, my friend Carla helped me with my presentation. It was a very exciting experience!

I would like to thank all the G_JT friends who were present at the meeting and listened with patience.¹



For those who love little animal icons, an awesome piece of news is that the gorgeous



package is now on CTAN.

Some duck fans use it to create the welcome party video² for Barbara Beeton: she retired in February, so she will have much time to spend on T_EX.SE, all the TikZlings are waiting for her!



Last but not least, on December, 17th, *The New York Times* dedicated an article to our Jedi Master, Prof. Donald E. Knuth, “The Yoda of Silicon Valley”.³ Unmissable reading for any T_EX fan, quack!



T_EX.SE friends are always very attentive in finding typos or inaccuracies in my articles.

The user ‘TeXnician’ pointed out there is a `text dept` instead of `text depth` in Figure 2 of *The Morse code of TikZ* in TUGboat 39:1.

About *Formatting posts* in TUGboat 39:3, marmot remarked that, on a Mac, shortcuts are usually done with the command key, even though `Ctrl` also works.

¹ For more information: <https://www.guitex.org/home/guit-meeting-2018>.

² <https://vimeo.com/315852862>.

³ <https://www.nytimes.com/2018/12/17/science/donald-knuth-computers-algorithms-programming.html>.

I would like to thank them both.



This time I will show you how to process text files to automatically obtain professional L^AT_EX tables. As we will see, it is straightforward and convenient.

But let us talk about the upvoting distribution by topics on T_EX.SE, first.

2 “Cinderella” topics

Not all T_EX.SE tags have the same level of popularity.

Since I was curious to know which were the “Cinderella” topics, that is, less valued but not less worthy, I asked my friend Carla to post a question on Meta⁴ for me.

We got a gorgeous answer by moewe, with a very detailed and accurate analysis of the vote distribution. His results are somehow surprising, quack!

First, there is a general feeling that TikZ related posts receive, on average, many more upvotes than others. Looking at the data, this is not entirely true.

Taking into account only the tags with more than 5,000 answers, there are some which perform better than TikZ, such as `symbols`, `macros`, and `math-mode`. Extending to tags with more than 2,000 answers, the absolute top topic is `tex-core`.

Looking at the “Cinderella” ones, meanwhile, moewe found out that `table-of-contents`, `floats`, and `tables`, among the most frequent tags, perform worse than other topics.

This is quite unexplainable because these are the first things beginners found “strange”, compared with ordinary word processors. Maybe the posts are trivial or focused on such peculiar problems that they do not matter for other users.

Extending the sample, also `bibliographies` and `LyX` can keep the seven dwarfs company.

The low reputation average of the latter is not a surprise. Many T_EXnicians do not like the WYSIWYG/M (What You See Is What You Get/Mean) philosophy adopted by this tool.

I do not think the idea is wrong *a priori*. I used this tool in the past; I gave up only because my LyX files were so full of ERT (Evil Red Text) that it was more convenient to write them in L^AT_EX directly.

As for `bibliographies` (especially `BIBLATEX`), maybe the few upvotes are due to the peculiarity of the questions, which specifically concern the problem of a single OP and are often not useful for others. Moreover, we have few BIBL^AT_EXperts (among which

⁴ <https://tex.meta.stackexchange.com/questions/7977/poll-which-are-the-cinderella-topics>.

moewe is undoubtedly the top one) who can fully understand and appreciate these answers.



One of the “Cinderella” tags is `csvsimple`. It has only 2.8 votes per answer, on average, and more than 8% of answers with no votes at all. Since I think this package is useful and I would like to make it more popular, I have dedicated the current Quack Guide to it.

3 Quack Guide No. 4

How to easily process text files to get L^AT_EX tables

We all know we can get beautiful tables with L^AT_EX, but typing them can be boring and error-prone.

We often have our data processed by another tool which yields a file in some text-based format (`.csv`, `.dat`, `.tex` or similar), and we only need to transform it into a L^AT_EX table.

There are some packages for this purpose, for example, `pgfplotstable` and `datatool`. I will show you `csvsimple`, which is the simplest one (that is the reason for its name, quack!) but sufficient for basic usage.

Suppose we have the CSV file `test.csv`, which contains the following data (any resemblance to real persons/ducks is purely coincidental):

test.csv

```
Name,Surname,Height,Gender
Paulinho,van Duck,.4,M
Paulette,de la Quack,.35,F
Enrichetta,Pescatore,1.80,
Henry,Gregory,,M
```

Pay attention to the missing data: the comma (or, in general, the separator) is needed, see, for example Enrichetta’s “Gender” or Henry’s “Height”.

With this simple code:

```
\documentclass{article}
\usepackage{csvsimple}
\usepackage{booktabs}
\begin{document}
\csvautobooktabular{test.csv}
\end{document}
```

we can already get a passable result, as shown in Table 1.

I used `\csvautobooktabular` because my table fits on one page; for multiple-page tables, there is `\csvautobooklongtable`.

Table 1: Result of `\csvautobooktabular` applied to our comma-separated test file with header line.

Name	Surname	Height	Gender
Paulinho	van Duck	.4	M
Paulette	de la Quack	.35	F
Enrichetta	Pescatore	1.80	
Henry	Gregory		M

As you may note, `csvsimple` does not itself load `booktabs` nor `longtable`; hence you have to load them separately.

The macros that do not need `booktabs`, that is `\csvautotabular` and `\csvautolongtable`, produce tables with vertical lines. However, since T_EX-nicians love vertical lines in tables as much as chairs covered with cactus, I recommend not using them, quack!



Of course, we can improve our table.

For example, suppose we want to correctly align the numeric data and change their heading, adding the unit of measure, and also centering the “Gender” column and swapping it with column “Height”.

All that can be done by putting a `\csvreader` within a `tabular` environment:

```
\documentclass{article}
\usepackage{csvsimple}
\usepackage{booktabs}
\usepackage{siunitx}
\usepackage{makecell}

\begin{document}
\begin{tabular}{c}
\toprule
Person/Duck & Gender & 
{\makecell{Height\ (\si{\metre})}}\ \\
\midrule
\csvreader[head to column names,
late after line=\\]{test.csv}{%
{\Name\ \Surname & \Gender & \Height}
\bottomrule
\end{tabular}
\end{document}
```

or, directly, with the appropriate options in the `\csvreader` command (in the following I will show only the `\csvreader` commands; the rest of the code is the same as before):

Table 2: A table created with `\csvreader` applied to our comma-separated test file with header line.

Person/Duck	Gender	Height (m)
Paulinho van Duck	M	0.40
Paulette de la Quack	F	0.35
Enrichetta Pescatore		1.80
Henry Gregory	M	

```
\csvreader[
  tabular={
    lcS[table-format=1.2,round-mode=places]},
  table head={\toprule
    Person/Duck & Gender &
    {\makecell{Height\ (\si{\metre})}}\
    \midrule},
  head to column names,
  late after last line=\bottomrule,
]{test.csv}{%
  {\Name\ \Surname & \Gender & \Height}}
```

Please note that I have also created a unique column with name and surname, simply by using:

```
\Name\ \Surname
```

The output of the two previous examples is the same, shown in Table 2.

The general syntax of the command is `\csvreader[<options>]{<file name>}%`
`{<assignments>}{<command list>}`

where *<file name>* is the name of your text file and *<command list>* is executed for every line.

The *<options>* can provide instructions for the table formatting, as follows.

With `tabular=<table format>` you can specify the column types you prefer.

In `table head=<code>` you can insert the code for the table headings.

`late after line=<code>` is the code to be executed after processing a line of the input file;⁵ if the option `tabular` is present, it is set to `\` automatically. Analogously, `late after last line=<code>` is executed after processing the last line of the file.

When `head to column names=true|false` is set to `true` (the default), the entries of the header line of the input file are used automatically as macro names for the columns.

This option cannot be used if the header entries contains spaces, numbers or special characters, because only letters can be used in L^AT_EX macro names. If this is the case, you can set `head to column`

⁵ To tell the truth, it is a bit more complicated than this, but for our purpose, we do not need to be fussy.

`names=false` and reference the columns with the `csvreader` macros: `\csvcoli` (`coli` means the first column; roman numerals are used since arabic numerals cannot appear in L^AT_EX commands), `\csvcolii`, `\csvcoliii`, and so on:

```
\csvreader[
  tabular={
    lcS[table-format=1.2,round-mode=places]},
  table head={\toprule
    Person/Duck & Gender &
    {\makecell{Height\ (\si{\metre})}}\
    \midrule},
  head to column names=false,
  late after last line=\bottomrule,
]{test.csv}{%
  {\csvcoli\ \csvcolii & \csvcoliv &
  \csvcoliii}}
```

Alternatively, you can take advantage of the *<assignments>* parameter, giving a customized macro name to the column entry, with *<name>*=*<macro>*, where *<name>* is the arabic number of the column (or the entry from the header, if you want to use another macro to identify the column):

```
\csvreader[
  tabular={
    lcS[table-format=1.2,round-mode=places]},
  table head={\toprule
    Person/Duck & Gender &
    {\makecell{Height\ (\si{\metre})}}\
    \midrule},
  head to column names=false,
  late after last line=\bottomrule,
]{test.csv}{Name=\myn, 2=\mys, 3=\myh,
  4=\myg}%
  {\myn\ \mys & \myg & \myh}}
```

It may be that your text file has no header line giving the field names at all. In this case, we can use `head=false` or its abbreviation `no head` (this option cannot be used with `\csvautotabular` or similar automated commands).

Let us see an example processing a second text file which has no header line and semicolons instead of commas as separators:

testnohead.csv

```
van Duck, Paulinho;.4;M
de la Quack, Paulette;.35;F
Pescatore, Enrichetta;1.80;
Gregory, Henry;;M
```

Table 3: A table created with `\csvreader` applied to our semicolon-separated test file with no header line; column names are defined using the `table head` option.

N.	Person/Duck	Gender	Height (m)
1	van Duck, Paulinho	M	0.40
2	de la Quack, Paulette	F	0.35
3	Pescatore, Enrichetta		1.80
4	Gregory, Henry	M	

Please note that, since the separators are semicolons, we can have commas in the entry. Entries with commas would also be possible if the separators were commas, but in that case the entries must be surrounded by curly braces, for example:

```
{van Duck, Paulinho},.4,M
```

Generally, this can be done by the tool that produced the text file, or by any spreadsheet program.

The option `separator=<sign>` indicates the separator character in the file, the possible values are: `comma`, `semicolon`, `pipe`, and `tab`.

The following macro gives the result shown in Table 3.

```
\csvreader[
  tabular={
    clcS[table-format=1.2,round-mode=places]
  },
  table head={\toprule
    N. & Person/Duck & Gender &
    {\makecell{Height\ (\si{\metre})}}\ \
    \midrule},
  nohead,
  separator=semicolon,
  late after last line=\ \ \bottomrule,
]{testnohead.csv}{1=\myn, 2=\myh, 3=\myg}%
{\thecsvrow & \myn & \myg & \myh}
```

You can see that I have also used the convenient macro `\thecsvrow` to write the row numbers.



The above is plenty for basic usage, but the `csvsimple` package has many other features.

With `\csvset<{option list}>` you can set some options valid for all your `\csvreader` commands.

For example, if all your text files lack a header line and are separated by pipes, you can set this once for all as in the following, avoiding repeating them every time:

```
\csvset{
  nohead,
  separator=pipe
}
```

`\csvstyle<{style name}><{option list}>` allows you to create any customized style you need.

For example, you could create `mystyle`:

```
\csvstyle{mystyle}{
  tabular={
    clcS[table-format=1.2,round-mode=places]
  },
  table head={\toprule
    N. & Person/Duck & Gender &
    {\makecell{Height\ (\si{\metre})}}\ \
    \midrule},
  nohead,
  separator=semicolon,
  late after last line=\ \ \bottomrule}
```

and then conveniently use it in your `\csvreader` commands:

```
\csvreader[
  mystyle
]{testnohead.csv}{1=\myn, 2=\myh, 3=\myg}%
{\thecsvrow & \myn & \myg & \myh}
```

There are also possibilities of filtering and sorting; further, you can use `\csvreader` not only for tables but also for any repetitive text, when the only things that change are the data in the text file. I will not discuss these features here, but I recommend reading the package documentation [1] for all the information.

4 Conclusions

I hope you enjoyed my explanation, and if you have any problem in processing a CSV file, remember:

Ask van Duck for a quack solution!

References

- [1] Thomas F. Sturm. *The csvsimple package. Manual for Version 1.20 (2016/07/01)*. <https://ctan.org/pkg/csvsimple>.

◇ Herr Professor Paulinho van Duck
Quack University Campus
Sempione Park Pond
Milano, Italy
paulinho dot vanduck (at) gmail dot com

No hands — the dictation of \LaTeX

Mike Roberts

Abstract

This article gives a brief introduction to a combination of open source extensions to Dragon Professional Individual dictation software which allow for relatively easy dictation of \LaTeX syntax and mathematical formulae.

While the primary use case is for people with disabilities which prevent them from typing (repetitive strain injuries caused by regular computer use are surprisingly common), dictation may provide a realistic alternative for normal users as well.

1 Introduction

1.1 Me

As an Economics undergraduate with a spinal injury which precludes me from using a keyboard, I've been using dictation software throughout my degree for all exams, assignments and essays. This article will focus on the workflow I've developed and the tools that I use for dictating \LaTeX documents and mathematical formulae relatively painlessly.

First, though, I will briefly describe the problem space: what would ideally be made possible by a dictation system and the limitations of what is currently available. I will then describe the technical details and user experience of my setup.

1.2 Dictation software

If you are totally unfamiliar with dictation software, the first thing I should say is that broadly speaking it works well. With a good microphone, in a quiet room, speaking clearly, it's not unreasonable to expect in excess of 99% accuracy when dictating full sentences.

The leader in the voice recognition industry has for many years been Nuance (though with the current rate of progress in machine learning their technological lead may soon evaporate). Their product, Dragon, is marketed mainly for corporate, medical and legal document preparation and works excellently when dictating into Microsoft Word. For completing the full range of academic work without a keyboard though, this basic dictation functionality is necessary but not sufficient, and for those with disabilities who cannot use a mouse, navigating a graphical interface to access formatting options is an active hindrance to getting things done.

For dictating mathematics, another commercial product is available — MathTalk [3] is the industry standard for mathematical dictation (the client list on their website includes the Department of Defense

and Federal Aviation Authority, as well as many universities). When I first tried it I was very disappointed and quickly found it to be practically unusable. Despite costing over \$300 it is incredibly slow to recognise and execute commands, and will not accept more than two or three commands at a time. This can easily be seen by watching any of the videos on their website. Even with extra time granted, the chances of being able to do well in exams when you have to wait half a second between every character are minimal.

1.3 The problem

There is certainly room for improvement then, both for normal documents and for mathematics. An ideal dictation system of this kind should have a number of features which are lacking from the commercial offerings detailed above. Firstly, it should be able to interpret commands as fast as the user can say them, without the need to wait for output to appear or to pause between commands. Secondly, it should be customisable so that users can modify and add to their grammar to suit it to their own needs. Finally, of course, it should as far as possible be free and open source.

2 Implementation

As I finish my degree and after a lot of experimentation I have settled on a solution which is as close to this ideal as I can imagine. It can be used for almost anything, can interpret commands as quickly as they can be dictated, and is modifiable and extensible.

2.1 Building blocks

Although it is somewhat expensive, Dragon [4] is by far the best speech recognition engine currently available, and while its built-in tools for creating custom commands are limited, it is thankfully hackable. This is done using Natlink [2], a free tool originally created in 1999 by Joel Gould — then working at Dragon — which allows for custom command sets written in Python 2.7 to be imported into Dragon.

These custom commands are defined using an open source Python library called Dragonfly [1], which simplifies the process of creating grammars and provides easy access to frequently used functionality like the typing of text and the execution of keystrokes.

Together, these elements — Dragon, Natlink and Dragonfly — allow for any combination of keystrokes or Python scripts to be mapped to voice commands which can be interpreted and executed fluidly and with only minimal delay. While these tools have so far primarily been used to enable voice programming,

they can easily be repurposed for voice-enabling virtually anything.

2.2 Mathfly

Mathfly [5] is my own contribution, and comprises command sets for dictating raw \LaTeX as well as using WYSIWYG editors like LYX .

Within Mathfly, commands are organised into modules, each with a different purpose, which can be enabled and disabled at will. For basic operations like creating a new file there are also context specific commands which will only be recognised when a particular program is active.

To provide usability for non-programmers, predictable and common structures (like the begin, end tags in \LaTeX) are hardcoded in Python while the lists of options themselves are defined in plain text configuration files which can all be opened and added to with voice commands.

3 \LaTeX

\LaTeX represents an obvious and favourable alternative to dictating into word processing software for a number of reasons. Writing everything in plain text means that entire documents can be produced by replicating keystrokes, without ever having to navigate an awkward GUI. This provides the ability to automate fiddly tasks like the creation of tables, insertion of images and organisation of references — a major win for those without the use of a mouse.

Dictating using Mathfly’s \LaTeX module is intended to be as intuitive as possible and to work largely as one would expect it to. Most commands consist of a memorable prefix, which helps to avoid over-recognition during dictation, followed by the name of the desired item.

3.1 Basic commands

For example, saying “begin equation” produces:

```
\begin{equation}
\end{equation}
```

Similarly, “insert author” and “insert table of contents” produces `\author{}` and `\tableofcontents`, respectively, and “use package geometry” will produce `\usepackage{geometry}`.

\LaTeX commands can often be a little cryptic and non-obvious. For example, to create a bulleted list, you need:

```
\begin{itemize}
```

This is reasonably memorable once you have used it a few times, but is not easily guessable, especially for those of us living in countries which resist the encroachment of the letter Z.

Mathfly attempts to make things as easy as possible in cases like these by often providing multiple voice commands for the same thing. In this case, “begin itemize”, “begin list” and “begin bulleted list” will all produce an itemize environment.

3.2 Mathematics

By default, all mathematical symbols are prefixed with “symbol”, so “symbol integral” produces `\int`, but there is also a mode specifically for dictating symbols which does not require the prefix. Thus in mathematics mode, “sine squared greek theta plus cosine squared greek theta equals one” produces:

```
\sin ^{2} \theta +\cos ^{2} \theta =1
```

that is,

$$\sin^2 \theta + \cos^2 \theta = 1 \quad (1)$$

3.3 Templates

For including larger sections of text, or sections which don’t fit into any of the predefined commands, there are templates — arbitrary strings which are pasted with a voice command. For example, the command “template graphic” pastes:

```
\begin{figure}[h!]
\centering
\includegraphics[width=0.8\textwidth]{
\caption{
\label{
\end{figure}
```

Not only does this save a lot of time and repetition, but as a novice user it is useful to be able to outsource the task of remembering what settings you like to use and how common command blocks are constructed.

3.4 Configuration

As I mentioned above, it is easy for users to add to the available commands by modifying the configurations files. The command definitions for the \LaTeX module look like this:

```
[environments]
"equation" = "equation"
...
[command]
"author" = "author"
...
```

and can be easily added to or modified.

3.5 Scripting

There are also intriguing possibilities for the integration of Python scripting. I’ve only scratched the surface so far, but to give an example I can highlight the title of a book or paper, say “add paper to

bibliography” and a script will run which searches Google Scholar for the title and appends the resulting BIB_TE_X citation to my .bib file.

4 WYSIWYG mathematics

For technical homework assignments and especially exams, formatting is of far less importance than getting what you know onto the paper as quickly and easily as possible, so a what you see is what you get (WYSIWYG) editor makes more sense.

Mathfly includes grammars for both LyX, an open source L^AT_EX editor, and Scientific Notebook, a proprietary alternative which is often provided for free by universities. They both function similarly and allow for natural dictation of mathematical formulae with immediately visible output.

For example, the command

```
integral
one over x-ray
right
delta
x-ray
equals
natural logarithm
x-ray
plus
charlie
```

can all be interpreted in one go and will produce the desired output.

$$\int \frac{1}{x} dx = \ln x + c \quad (2)$$

The only deviation from natural speech is the requirement for a command (a right keypress) to signal the end of the fraction.

I don’t have any hard data comparing the speed of dictation like this to that of normal writing. I can say, though, that I am technically allowed 50% extra time for all exams but have never needed to make use of it, suggesting that the two methods are fairly comparable.

5 Limitations

The major limitations of dictation are currently not functional — it performs about as well as could reasonably be expected — but are related to platforms and compatibility. Dragon and Natlink are only available on Windows (with a limited and soon to be discontinued version of Dragon available for Mac OS X), so the only feasible way of using software like Mathfly on other operating systems is to run Dragon in a Windows Virtual Machine, using remote procedure calls to send instructions to the host.

The long-term prospects for a completely free and platform agnostic dictation and voice command framework are reasonably good, however. Dragonfly is under active development with the aim of integrating new speech engines like Carnegie Mellon University’s PocketSphinx and Mozilla’s DeepSpeech, although these have a fairly long way to go before they reach Dragon’s level of maturity and accuracy.

6 Getting started

Anybody interested can visit the Mathfly website [5], which contains links to the documentation, installation instructions and a few short video demonstrations. If you have any questions or requests then feel free to email me, post in the Gitter chat room or on the GitHub issues page.

References

- [1] C. Butcher. Dragonfly, 2007. pythonhosted.org/dragonfly
- [2] Q. Hoogenboom. About Natlink, Unimacro and Vocola. qh.antenna.nl/unimacro
- [3] mathtalk.com. MathTalk — speech recognition math software. mathtalk.com
- [4] nuance.com. Dragon — the world’s no. 1 speech recognition software (nuance UK). nuance.com/en-gb/dragon.html
- [5] M. Roberts. Mathfly — dragonfly/caster scripts for dictating mathematics fluidly, 2019. mathfly.org

◇ Mike Roberts
 mike (at) mathfly dot org
mathfly.org

Nemeth braille math and \LaTeX source as braille

Susan Jolly

Abstract

This article, which is dedicated to the late \TeX expert Prof. Eitan Gurari [13], introduces braille math for sighted persons unfamiliar with braille. Braille systems represent print in one of two ways: either by transcription or by transliteration. Most braille systems, including the Nemeth system for mathematics, employ shorthand, markup, meaningful whitespace, context-sensitive semantics, and other strategies to transcribe general printed material to a six-dot braille format that accommodates the special requirements of tactile reading. Transliteration, by contrast, is limited by the small number of braille cells and is typically only used for representing plain text ASCII files such as \LaTeX source and computer code.

This article argues that while reading and writing mathematics as \LaTeX source transliterated to braille is possible for facile braille readers who have eight-dot refreshable braille displays and are able to learn \LaTeX , it is not an appropriate general solution for making mathematics accessible to braille users. Tactile reading of transliterated \LaTeX source is no different from visual reading of \LaTeX source. On the other hand, tactile reading of math transcribed using the Nemeth system is the tactile analog to visual reading of *rendered* math. Simulated braille is used to illustrate this for the benefit of sighted readers.

1 Braille cells

Most sighted persons reading this article are probably at least somewhat familiar with the punctiform appearance of braille and are aware that the dots need to be physically raised for tactile reading as opposed to visual reading. The individual braille characters, which are usually referred to as braille cells, are officially designated as Unicode Braille Patterns. Technically speaking, the braille cells aren't characters per se but rather *symbols* assigned meanings by a braille system. There are currently more than 150 different braille systems. The majority represent natural languages but there are several systems for mathematics and for other specialized material, including music and chess.

There are two forms of braille: six-dot and eight-dot. The standard six-dot patterns are comprised of three rows of two dot positions each and the eight-dot ones are comprised of four such rows. There are thus 63 different six-dot braille patterns with at least one dot and 255 eight-dot ones. Unfortunately

Unicode does not treat the six-dot cells as separate character codes, but simply as the subset of the eight-dot patterns which happen to have both dot positions in their fourth row unfilled. This omission requires the use of additional information to avoid an extra blank line when embossing a braille file encoded as Unicode Braille that is intended as standard six-dot braille and the use of a custom simulated braille font to avoid misalignment when typesetting Unicode Braille intended as six-dot braille.

2 Transliteration and computer braille

Conversion of print to braille is usually done with braille systems using transcription, in order to accommodate the special requirements of tactile reading, including those described later in this article. This section describes conversion done with transliteration.

Transliteration is of limited practical use except for conversion of plain text ASCII files to braille. Since there are 94 ASCII keyboard characters (excluding space) and only 63 six-dot braille cells, one-for-one transliteration of plain text files isn't possible with six-dot braille. Six-dot transliteration of plain text files thus requires some sort of braille system that uses both single-cell and two-cell (prefix-root) symbols. One-for-one transliteration of plain text is of course possible with the use of eight-dot braille.

General eight-dot braille systems aren't widely used since tactile recognition of all 255 eight-dot cells isn't feasible for the majority of tactile readers. Nonetheless, many refreshable braille displays have eight-dot cells and incorporate built-in support for an ad hoc one-for-one eight-dot transliteration of the ASCII characters. This transliteration is known as computer braille because of its usefulness for representing computer code.

Computer braille adds 31 cells with a dot in the left columns of their fourth rows to the 63 standard six-dot cells. Twenty-six of these additional cells simply represent capital letters by adding the extra dot to the six-dot pattern for the corresponding small letter. Several options for the remaining five are in use. Because computer braille is a limited extension to six-dot braille it, in contrast to general eight-dot braille, is feasible for tactile reading.

Since computer braille specifies a one-for-one transliteration of all 94 ASCII keyboard characters it thus provides a method for braille users to directly read and write any ASCII-based plain text including classic \LaTeX source. The advantage is that computer braille transliteration doesn't require special software to translate from print to braille or to back-translate from braille to print. The disadvantage is that \LaTeX source and many other plain text source files are

primarily intended to be rendered for visual reading, not to be read directly.

3 Nemeth braille math

The well-known six-dot Nemeth braille system, “The Nemeth Braille Code for Mathematics and Science Notation”, is an outstanding example of a braille system. It was developed by Dr. Abraham Nemeth (1918–2013), a congenitally blind American mathematician who became a facile braille reader as a young child and was a math professor at Wayne State University for some 30 years. It had been in use for a number of years prior to the adoption and publication of the current official version in 1972. A PDF facsimile of the 1972 book is now available for download [1]. This book incorporates a thoughtful guide to numerous issues for converting print math to braille so as to retain all the essential information while avoiding extraneous print-specific details.

Nemeth braille was designed with a thorough understanding of both mathematics and the requirements for efficient tactile reading as outlined in the next section. Nemeth is a complete system for representing text and mathematics and includes formatting specifications. Since it was designed long before the use of digital media, it was originally intended for embossed braille transcriptions of math textbooks and other STEM material as well as for direct entry of math by braille users. The Nemeth system for mathematics has stood the test of time and is used not only in the United States but in numerous other countries, including India [12]. It is used with refreshable braille displays and, like \LaTeX , its “math mode” is used for representing individual math expressions. There is also an equivalent form for spoken math called MathSpeak. MathSpeak is a unique method of speaking math since it is the only method of speaking math that supports dictation of math so it can be written correctly in either print or Nemeth braille [5].

Although Nemeth braille is a complete system, the United States, along with other English-speaking countries, recently adopted the somewhat controversial Unified English Braille (UEB) system [6]. In the U.S. it replaces both Nemeth braille and the prior system for English braille. The UEB math specification is awkward for a number of reasons, one being the use of the same braille cells for the decimal digits and for the letters a–j. The negative reaction to UEB math in the United States is significant enough that many U.S. states allow for Nemeth math, tagged with switch indicators, to be used for transcribing math content together with UEB used only for non-math text [2, lesson 3.8].

The next several sections of this article provide background for understanding the pros and cons for braille users of using computer braille to read and write mathematics as transliterated \LaTeX source, versus using Nemeth braille for that purpose.

4 Tactile reading

This section describes some of the aspects of tactile reading that are taken into account when developing braille systems such as Nemeth braille. Note that braille systems are linear because tactile readers can’t easily sense the relative vertical positions of the braille cells.

The rate of tactile reading, rather like the rate of typing on a standard keyboard, depends more on the number of braille cells than on the number of words. Braille systems that have been designed to minimize the number of braille cells can thus be read more efficiently.

The rate of tactile reading is also affected by particular dot patterns. For example the seven six-dot braille cells that only have dots in their right column are easier to recognize when used as markup or indicators affecting a subsequent braille cell or cells and are typically used for this purpose. The cell with a single dot at the bottom of its right-hand column indicates that the following letter is capitalized. Nemeth braille uses several others of these seven indicators to identify transliterations of Greek and other non-English alphabets.

Certain braille cells or sequences can be hard to distinguish from other possibilities. Distinguishing a lower braille cell, one which has filled dots only in its lower two rows, from the corresponding upper cell, which has the same dot pattern in its upper two rows, is an example. Nemeth uses the lower cells for the decimal digits but ensures that they are easily distinguished from the corresponding upper cells by requiring a preceding dot locator indicator, a cell with dots in all three rows, before a digit that would otherwise be at the start of a line or preceded by a space.

Braille systems can be easier to remember and recognize when they use tactile mnemonics related to the specific patterns of the braille cells. For example, the cell with dots in the upper two positions on its right is the superscript indicator and the one with dots in the lower two positions on its right is the subscript indicator. Another type of tactile mnemonics in Nemeth braille uses related dot patterns for related mathematical symbols. Examples are pairs of mirror-image braille symbols used for parentheses and for the less-than and greater-than symbols.

Finally, Nemeth uses braille-specific constructs to reduce the memory load for tactile comprehension of complex expressions. Nemeth, like \LaTeX (or MathML), naturally represents planar layouts, including fractions, in a linear manner. However, when an expression, such as a fraction with another fraction in its numerator, requires nested layout indicators of the same type, Nemeth adds an explicit indication of the nesting by prefacing outer layout indicators with additional markup to represent the order or level of nesting. (This is similar to the usefulness to visual readers of highlighting matching pairs of nested parentheses using different colors for each pair.)

In summary, braille systems like Nemeth braille are carefully designed to accommodate the special requirements of tactile reading. This is in contrast to computer braille transliterations of plain text files originally designed for other purposes.

5 Simulated braille example of Nemeth math

This section uses simulated braille to display the Nemeth braille translation of the well-known equation,

$$e^x = \int_{-\infty}^x \sum_{n=0}^{\infty} \frac{\lambda^n}{n!} d\lambda$$

This equation, chosen in part because it employs three different planar layouts, illustrates how Nemeth's elegant use of tactile mnemonics and other tactile considerations enhances the tactile readability and information content of the braille math.

First, here's a transliteration from Nemeth braille to standard ASCII Braille used for six-dot braille:

```
e^x .k $;- ,=^x" . ,s<n .k #0% ,=] ? .1^ n" /n&#d.1
```

This may look odder than other markup languages, but I've found it helpful and not too difficult to learn. Of course, since braille transcribed according to braille systems isn't one-for-one with print and also because the same braille cells typically have different semantics in different contexts, an ASCII Braille transliteration is simply a print equivalent of the braille, not a backtranslation of the braille to print. Nonetheless, the letters and digits and some special characters can be read directly in transliterated Nemeth math. Also, some of the other print characters in ASCII Braille are used because their glyphs resemble the dot patterns of the corresponding braille cells.

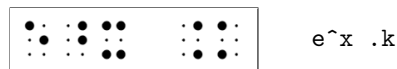
The Nemeth braille for the example uses 40 braille cells and 4 spaces. The number of braille cells is approximately 60 percent of the number of print characters in the corresponding \LaTeX source.

5.1 Simulated display of Nemeth braille

The simulated braille for this expression is displayed below in five segments to make the descriptions easier to follow. Note that the standard six-dot simulated braille font used here has shadow dots in the unfilled positions. Shadow dots are intended to make visual reading easier although one may need to take care not to let the shadow dots obscure the dot patterns of the corresponding tactile braille cells.

5.1.1 First segment

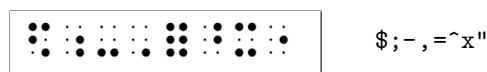
Now let's try to see how the Nemeth braille for $e^x =$ from the equation above would be experienced by tactile readers. It looks like this:



The first and third cells are the standard cells for the lower case letters e and x so are nothing new for a braille reader. The second cell, described previously, indicates that the following expression is a superscript. This superscripted expression is terminated by default by the space always required before comparison symbols. The two-cell symbol for an equals sign purposely resembles a print equals sign, as Dr. Nemeth believed that such similarities helped communication between braille readers and their sighted peers and teachers.

5.1.2 Second segment

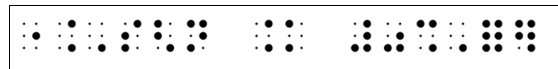
The second segment represents $\int_{-\infty}^x$:



The first cell in this segment, which somewhat resembles an integral sign, is the braille symbol for a single integral. The second cell is a subscript indicator with its argument terminated by default by the superscript indicator also used in the first segment. The three cells following the subscript indicator thus represent the subscripted expression. The minus sign is obvious. It is followed by the indicator cell with one dot in its lower right, familiar to braille readers from its use to indicate capital letters in six-dot systems. This indicator is also used in Nemeth math to indicate that it together with the next non-alphabetic cell is a special symbol; here, the braille cell used for infinity, resembling a rotated print infinity symbol. The cell following the superscript indicator is the symbol for x , also used above in the first segment, and the last cell is required to explicitly terminate the superscripted expression since the next item in the expression isn't a space.

5.1.3 Third segment

The third segment is a symbol decorated with, as termed in braille, an underscript and overscript, $\sum_{n=0}^{\infty}$:



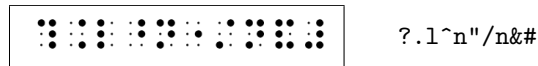
". ,s<n .k #0%,=]

The first cell in this segment, with just one dot, is the indicator specifying that the next item is decorated. The fourth cell is the braille cell for the letter *s*, which is here transliterating a capital Greek sigma per its two preceding indicators. This is followed by the Nemeth underscript indicator and then the expression for $n = 0$ which uses the standard cell for the letter *n*, the same space-delimited symbol for an equals sign used in the first segment; the number sign dot locator described in Section 4, which is required because the following digit would otherwise be preceded by a space; and then the lower cell for the digit zero. The zero is followed by the Nemeth overscript indicator and then the same two-cell symbol for infinity used in the second segment. The last cell is the required Nemeth terminator for any layout using one or more underscripts and/or overscripts.

This is a case where the Nemeth math, which is intended to represent print presentation in a consistent manner, is especially lengthy in comparison with the compact print rendering. It might be desirable to develop more informative print shorthand rather than replicating print presentation for common expressions that use underscripts and overscripts. For example, since summation is essentially a function application, a custom string like “sumnzi” could be added to the function name abbreviations already recognized by Nemeth braille. This would reduce the number of braille cells and spaces for this segment from 17 to 7 counting the extra space required to separate a Nemeth function reference from its argument and would thus be a reduction of about 25% in the number of braille cells in the entire expression.

5.1.4 Fourth segment

The fourth segment of the formula above is the fraction $\frac{\lambda^n}{n!}$, which uses the Nemeth simple fraction layout indicators:

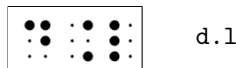


The first cell in this segment, which resembles an upside-down print L, is a strong tactile shape used as the fraction start indicator. The last cell, which is used as a dot locator in other contexts, is another strong tactile shape that is here used

as the fraction end indicator. The cell with two dots that resembles a print forward slash separates the numerator from the denominator. The third cell is the letter *l*; you shouldn't have too much trouble reading the numerator since the other four cells have already been encountered. The letter *n* in the denominator is followed by the one-cell Nemeth braille symbol for a factorial sign.

5.1.5 Fifth segment

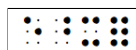
The fifth and final segment, $d.l$, is simply $d\lambda$:



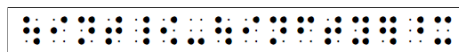
The first cell is the letter *d*. The remaining two cells are the symbol for lambda also used in the fourth segment.

5.2 L^AT_EX source as simulated braille

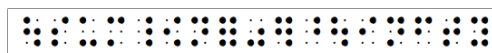
I hope that these descriptions have allowed you to appreciate how both the tactile form and the braille symbols specified by Nemeth braille supply information to tactile readers. Here, for contrast, is the corresponding ASCII Braille transliteration of the L^AT_EX source for each segment:



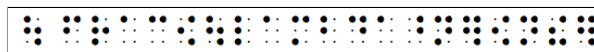
e^x=



\int_{-\infty}^x



\sum_{n=0}^{\infty}



\frac{\lambda^n}{n!}



d\lambda

6 Conclusion and future work

Here is some good news: two hardware issues for real-time access to braille have recently been addressed. First, new hardware designs have resulted in significantly cheaper single-line braille displays such as the six-dot BrailleMe [11]. Second, and of special importance for braille math, is the newly available six-dot Canute braille display which is not only low in cost but also the first multi-line refreshable braille display [3].

An urgent need for future work is accurate and free automated backtranslation from braille math to

print math. Currently available applications, most of which are not free, are problematic and students typically require their often unavailable itinerant braille teachers to interpret their braille work for their classroom teachers. Addressing this issue is a current goal of the Euromath project [4]. High school and college students sometimes resort to learning to read and write \LaTeX math as a result of poor support for braille math. In my opinion their time as students would be much better spent on improving their mathematical ability. In any case, \LaTeX source is not an especially convenient basis for manipulating math.

A possible starting point for providing automated backtranslation is the beta version of my free and open source BackNem 3.0 app for accurate backtranslation of Nemeth math to MathML, as demonstrated by several samples [9]. This app, which is based on the ANTLR 4 parser generator, is to my knowledge the first use of parsing technology for backtranslation of braille to print [7]. One valuable feature of ANTLR 4, which is especially important in educational contexts, is that its parsers can recognize input errors, provide optional developer-supplied error messages and, unlike other parsers, continue processing despite encountering input errors.

Future work needed to support the claims in this article includes development of a software system for real-time conversion of \LaTeX source to six-dot braille mathematics designed for integration with screen readers and other applications. The difficulties of direct conversion of \LaTeX to other formats is well-known. A two-step process that first converts \LaTeX to MathML with one of the currently available applications and then converts MathML to braille math is a more viable approach. This second step is straightforward for Nemeth math due to similarities between it and MathML, and a beta version of my MML2Nem app is available for consideration of a new approach [8, 10]. The need for real-time translation is especially critical for education due to the recent dramatic increase in the use of electronic information in this context.

Finally, I should point out that I'm not in a position to develop the needed software nor to provide the infrastructure necessary to test, distribute, or maintain software. I am however very glad to volunteer to help other developers of open source braille software as well as to answer questions about braille mathematics.

References

- [1] AAWB-AEVH-NBA, Advisory Council to the Braille Authority, Louisville, KY. *The Nemeth Braille Code for Mathematics and Science Notation. 1972 Revision*, 1987. www.brailleauthority.org/mathscience/nemeth1972.pdf
- [2] American Printing House for the Blind, Inc., 1839 Frankfort Avenue, Louisville, KY. *Nemeth Tutorial*, 2015. nemeth.aphtech.org
- [3] Bristol Braille Technology CIC. *Electronic Braille: reimaged*, 2018. bristolbraille.co.uk
- [4] EuroMath Education Platform. EuroMath project, 2017. www.euromath.eu/wp-content/uploads/2018/12/EuroMath_Presentation.pdf
- [5] gh, LLC. 2004 MathSpeak initiative, 2004. www.gh-mathspeak.com
- [6] C. Gray. Call to action: Prevent educational harm to Braille readers, 2014. all4braille.org/C1002.htm
- [7] S. Jolly. Using ANTLR 4 for braille translation, 2011. github.com/SusanJ/BasicUEB/wiki
- [8] S. Jolly. Positive impacts of EPUB 3: MathML and Braille mathematics, 2012. www.dotlessbraille.org/mathmlandbraille.htm
- [9] S. Jolly. Samples of BackNem 3.0 output, 2018. github.com/SusanJ/Baknem/wiki/Samples
- [10] S. Jolly. Translation of MathML to Nemeth Braille, 2018. github.com/SusanJ/MML2Nem
- [11] National Braille Press. Braille Me, n. d. www.nbp.org/ic/nbp/BRAILLE-ME.html
- [12] Overbrook School for the Blind; International Council for Education of People with Visual Impairment; The Nippon Foundation. *Mathematics Made Easy for Children with Visual Impairment*, 2005. www.hadley.edu/Resources_list/Mathematics_Made_Easy_for_Children_with_Visual_Impairment.pdf
- [13] D. Walden. Profile of Eitan Gurari (1947–2009). *TUGboat* 30(2):159–162, 2009. tug.org/TUGboat/tb30-2/tb95gurari.pdf

◇ Susan Jolly
 120 Dos Brazos
 Los Alamos, NM 87544, USA
 easjolly (at) ix dot netcom dot com

Both \TeX and DVI viewers inside the web browser

Jim Fowler

Abstract

By using a Pascal compiler which targets WebAssembly, \TeX itself can be run inside web browsers. The DVI output is converted to HTML. As a result, both \LaTeX and $TikZ$ are available as interactive input languages for content on the web.

1 Introduction

Many people would like to make technical material (often written in \TeX) available on the World Wide Web. Of course, this can be done via web pages, but for mathematical expressions, HTML and MathML produce inferior results. Consequently, many users rely on client-side tools like MathJax [1] to provide beautiful rendering for content in math mode.

There is also a need to go beyond math mode. How might one render a $TikZ$ [14] picture on the web? In the past, this might have been done with $\TeX4ht$ [8] to convert a $TikZ$ picture to SVG. This article describes the basis of a new method, $TikZJax$ [3], which, like MathJax, is client-side, performing its conversions in the client's browser. When the $TikZJax$ JavaScript is run, any $TikZ$ pictures inside `<script type="text/tikz">` tags are converted into SVG images. $TikZJax$ is emphatically not a JavaScript reimplementaion of $TikZ$, but instead works by running $\varepsilon\text{-}\TeX$ itself inside the user's web browser; this copy of \TeX is provided to the browser with its memory already loaded with $TikZ$.

In short, \TeX has been ported to JavaScript. This article describes how we ported \TeX to the JavaScript-based environment of web browsers, and how we render the resulting DVI output in HTML. We hope that making \TeX itself available in the browser will open up many new possibilities.

2 A Pascal compiler targeting web browsers

\TeX was written in an era when computing resources were rather more constrained than today. Many of those constraints have returned within the JavaScript ecosystem, e.g., JavaScript is slower than native code and has limited access to persistent storage.

2.1 Goto is a challenge

To run \TeX in a web browser, we initially wrote a Pascal compiler targeting JavaScript. The main challenge is handling `goto` which is used fairly frequently in Knuth's code (especially since the Pascal of that era did not offer an early `return` from pro-

cedures and functions), and does not exist as such in JavaScript. However, JavaScript does support labeled loops, labeled breaks, labeled continues, and alongside a trampoline-style device it is possible to emulate in JavaScript the procedure-local `gotos` used in \TeX . There are a handful of cases in which a non-local `goto` is used by \TeX to terminate the program early, but early termination can also be handled in JavaScript.

Thus, it *is* possible to transpile Pascal to JavaScript. However, it turns out that running \TeX inside JavaScript is not particularly efficient!

2.2 WebAssembly

WebAssembly [9] provides a speedier solution. WebAssembly is a binary format for a stack-based virtual machine (like the Java Virtual Machine) which runs inside modern web browsers and is designed as a compilation target for languages beyond JavaScript. There is still no support for `goto`, but the same tricks with labeled loops that make `goto` possible in JavaScript again work in WebAssembly. Our compiler `web2js` [4] digests the dialect of Pascal code that \TeX is written in and outputs WebAssembly, which can then be run inside modern web browsers. We chose the “web” in `web2js` to evoke both `WEB` and also the World Wide Web.

WebAssembly, as it is currently implemented in web browsers, does not provide any high-level dynamic memory allocation; it is possible to resize the heap but nothing like `malloc` is provided. Given that \TeX also does no dynamic allocation, it's relatively easy to compile \TeX to this target.

Since we want to run \LaTeX in the browser, it is necessary to use a \TeX distribution which supports the $\varepsilon\text{-}\TeX$ extensions. So before feeding the Pascal source code to `web2js`, we `TANGLE` in the change file for $\varepsilon\text{-}\TeX$. Other change files are needed too. For instance, there is a patch to the Pascal code needed to get the current date and time from JavaScript.

Some additional JavaScript code is needed to support components missing in the browser. For instance, there is no filesystem in the browser, so the Pascal filesystem calls make calls to JavaScript which provides a fake filesystem. The terminal output of \TeX can be viewed by opening the “Web Console” in the web browser. Satisfyingly, when it is all working, the \TeX banner is visible right there.

2.3 Why Pascal? Why not C?

There are other approaches to getting \TeX to run well in a web browser. An older project, `texlive.js`, achieves this goal via `emscripten` [15], a C compiler which targets WebAssembly. The resulting website

enables client-side creation of a PDF, and so depends on a PDF viewer to see the result. S. Venkatesan [13] discussed this approach and the limitations of PDF output in particular.

2.4 Putting it all together

In the quest for better performance, the same tricks that \TeX used historically with format files and memory dumps can be reused in the web browser. The underlying theme is that the ecosystem of a web browser, and its limitations, is more similar to computing in the early 1980s than might have been easily believed.

As with \TeX version 3.0, we do not bother making a special `initex` version and simply allocate a large number of memory cells to a single version of \TeX . A program called `initex.js` then loads the initial \LaTeX format (with only some hyphenation data) and whatever piece of a preamble (e.g., `\usepackage{tikz}`) might be useful for the desired application. Then the WebAssembly heap is dumped to disk, just as would have been done with `virtex` historically. This produces a file, `core.dump.gz`, which is only a couple of megabytes (after compression).

Note that `initex.js` is executed on a machine that already has a complete \TeX distribution installed, such as \TeX Live. By loading packages and then dumping core on a machine with a complete distribution, it is not necessary to ship much in the way of support files to the browser.

On the browser, both the WebAssembly machine code and `core.dump.gz` are loaded, the dump decompressed, and execution begins again at the beginning of the \TeX code but this time with the previously dumped memory already loaded. As described in the \TeX 82 source code [11, Part 51, Section 1331], when \TeX is loaded in such a fashion, the `ready_already` variable is set in such a way as to shortcut the usual initialization, making this browser-based version of \TeX ready to receive input very quickly.

3 Rendering DVI in HTML

Running \TeX is only half the problem. To build a viewer for the output of \TeX , the easiest format to parse is DVI [6, 7]. A DVI file is just a series of commands which change the current position, place characters and rules on the page, change the current font, etc.

Some previous projects make it possible to view DVI files from within web browsers. For instance, `dvihtml` [12] uses DVI specials to appropriately tag pieces of the content so that they can be wrapped by appropriate HTML tags, similar to \TeX 4ht [8].

Other projects like DVI2SVG [5] translate DVI into SVG with a Java-based tool.

Our new tool is called `dvi2html` [2] and works somewhat differently. For starters, unlike DVI2SVG, our new tool is written in JavaScript (and mostly TypeScript) so it runs in the browser. It is used to read the output of $\varepsilon\text{-TeX}$, running in the browser, and output HTML in real-time.

3.1 Fonts

Why wasn't all this done years ago? One significant challenge was the state of "fonts" on the web. Conveniently, it is possible (and relatively easy with CSS) to load server-provided fonts. To support Computer Modern and the like, `dvi2html` presently relies on the BaKoMa TrueType fonts, but given their license, it would be good to generate fonts for the web following MathJax's technique.

It must be mentioned that while fonts can be loaded, the web ecosystem lacks a robust way to query metric information. So we still end up shipping the standard collection of `.tfm` files to the browser, all base64-encoded and placed into a single `.json` file. A significant portion of the code comprising `dvi2html` is designed to parse \TeX Font Metric files.

3.2 The challenge of the baseline

But selecting the appropriate typeface is not enough; an HTML viewer for DVI must also position the glyphs in the appropriate positions. This is sadly harder than it ought to be. Although HTML5 supports many methods for positioning text, it does not support positioning text relative to a specified baseline.

A solution to this is available precisely because of the previously loaded metric information. By knowing where the top of the glyph is relative to the baseline, we can use HTML to place the glyph in the correct position.

3.3 Streaming transformation

Instead of a monolithic converter, `dvi2html` is structured as a streaming transformer via asynchronous generator functions. In particular, an input stream is transformed into an object stream of DVI commands. Since many DVI commands come in a variety of lengths (i.e., one-byte, two-byte, three-byte, four-byte versions), this initial transformation collapses the variety of commands in the binary format to a single command.

Armed with a sequence of DVI commands, additional transformations can be applied. For instance, there is some overhead to placing a single glyph

Both \TeX and DVI viewers inside the web browser

on the page in HTML, so one transformer takes sequential SetChar commands from the DVI input and collects them into a single SetText command which can place a sequence of glyphs on the page at once.

The real benefit, though, to stream transformations is that the various transformations can be composed, with new transformations plugged in as desired. For instance, a package like `xcolor` will generate `\specials` with push color and pop color commands, and these can be processed by a single stream transformer which understands these color commands. Another composable transformer knows about raw SVG data and can appropriately emit such code into the generated HTML.

Finally, this sort of design will make it possible to compose new transformers for hitherto unimagined `\specials`. Most interestingly, such `\specials` could facilitate additional interactivity on the web in future versions.

4 Some next steps

The tools for running \TeX itself inside a browser are useful for more than `TikZJax`. For instance, these same tools make a “live \LaTeX editor” possible in which a user can edit \LaTeX source in a web page and view the resulting DVI without installing software and without relying on a cloud-based \LaTeX compilation service.

The Ximera platform provides `\answer` which creates answer blanks within mathematical expressions. For instance, `1 + 3 = \answer{4}` creates an equation in which the right-hand-side is an answer blank. It would be wonderful to add `\answer` to a copy of \LaTeX running in the browser.

Additional extensions to \TeX itself are possible, like a hypothetical `jsTeX` which would extend \TeX with the ability to execute JavaScript code, akin to `LuaTeX` [10]. The reader can imagine additional applications of this platform.

References

- [1] D. Cervone. MathJax: A platform for mathematics on the Web. *Notices of the AMS* 59(2):312–316, 2012. ams.org/notices/201202/rtx120200312p.pdf
- [2] J. Fowler. `dvi2html`. github.com/kisonecat/dvi2html, 2019.
- [3] J. Fowler. `TikZjax`. github.com/kisonecat/tikzjax, 2019.
- [4] J. Fowler. `web2js`. github.com/kisonecat/web2js, 2019.
- [5] A. Frischauf and P. Libbrecht. DVI2SVG: Using \LaTeX layout on the Web. *TUGboat* 27(2):197–201, 2006. tug.org/TUGboat/tb27-2/tb87frischauf.pdf
- [6] D. Fuchs. The format of \TeX ’s DVI files. *TUGboat* 1(1):17–19, Oct. 1980. tug.org/TUGboat/tb01-1/tb01fuchs.pdf
- [7] D. Fuchs. Erratum: The format of \TeX ’s DVI files. *TUGboat* 2(1):11–11, Feb. 1981. tug.org/TUGboat/tb02-1/tb02fuchszab.pdf
- [8] E. M. Gurari. \TeX 4ht: HTML production. *TUGboat* 25(1):39–47, 2004. tug.org/TUGboat/tb25-1/gurari.pdf
- [9] A. Haas, A. Rossberg, et al. Bringing the web up to speed with WebAssembly. *ACM SIGPLAN Notices* 52(6):185–200, 2017.
- [10] T. Hoekwater. `LuaTeX`. *TUGboat* 28(3):312–313, 2007. tug.org/TUGboat/tb28-3/tb90hoekwater-luatex.pdf
- [11] D. E. Knuth. *TeX82*. Stanford University, Stanford, CA, USA, 1982.
- [12] M. D. Sofka. \TeX to HTML translation via tagged DVI files. *TUGboat* 19(2):214–222, June 1998. tug.org/TUGboat/tb19-2/tb59sofka.pdf
- [13] S. K. Venkatesan. \TeX as a three-stage rocket: Cookie-cutter page breaking. *TUGboat* 36(2):145–148, 2015. tug.org/TUGboat/tb36-2/tb113venkatesan.pdf
- [14] Z. Walczak. Graphics in \LaTeX using `TikZ`. *TUGboat* 29(1):176–179, 2008. tug.org/TUGboat/tb29-1/tb91walczak.pdf
- [15] A. Zakai. Emscripten: An LLVM-to-JavaScript compiler. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, pp. 301–312. ACM, 2011.

◇ Jim Fowler
100 Math Tower, 231 W 18th Ave
Columbus, Ohio 43212
USA
fowler (at) math dot osu dot edu
<http://kisonecat.com/>

Markdown 2.7.0: Towards lightweight markup in \TeX

Vít Novotný

Abstract

Markdown is a lightweight markup language that makes it easy to write structurally simple documents. Existing tools for rendering markdown documents to PDF treat \TeX as a black box. In contrast, the Markdown package provides support for styling and typesetting markdown documents in \TeX , extending a \TeX ie's toolbox rather than forcing her to replace \TeX with a more limited tool.

Since its release in 2016, the package has received several important updates improving the functionality and user experience. In this article, I will reintroduce the package, and describe its new functionality and documentation.

1 Introduction

The primary strength of \TeX lies perhaps in its programming and typesetting capabilities, not its syntax. Outside mathematics, non-programmable markup languages such as Markdown [2] provide a gentler learning curve, improved readability, and effortless single-source publishing for an author.

Existing tools for rendering markdown documents to PDF, such as Pandoc [1, 3] and MultiMarkdown, have several important disadvantages, which I discussed in my previous article [5]. These disadvantages include black-boxing \TeX , inconsistent support for \TeX commands in markdown documents, increased complexity of maintenance, and the lack of support for online \TeX services such as Overleaf. The Markdown \TeX package [5] overcomes all of these.

In my previous article, I introduced version 2.5.3 of the Markdown package, which was plagued by several shortcomings: The package would not function correctly when the `-output-directory` \TeX option was specified, since the package interacts with an external Lua interpreter that is unaware of \TeX options. The package was also wasteful with system resources, clogging up the file system with converted markdown documents unless the user cleaned them up manually. The documentation of the package was complete, but provided little help to the non-technical user.

In this article, I introduce version 2.7.0 of the Markdown package, which tackles the above problems and introduces several new features, such as content slicing and the Lua CLI. In Section 2, I will show the new features. In Section 3, I will describe the new documentation of the package.

2 New features

Between versions 2.5.3 and 2.7.0 of the Markdown package, there was one important patch version, 2.5.4, and two important minor versions, 2.6.0 and 2.7.0. Version 2.5.4 added support for the \TeX option `-output-directory`, version 2.6.0 introduced the Lua command-line interface (CLI) and added support for the `doc` \LaTeX package [4], and version 2.7.0 introduced the user manual and content slicing.

In this section, I will show the new features. Although the package also supports plain \TeX and Con \TeX t, all examples are in \LaTeX for ease of exposition.

2.1 Setting the output directory

\TeX provides the `-output-directory` option, which changes the working directory for the \TeX document. This allows the user to redirect auxiliary files to a separate location. However, any external programs executed using the `\write18` mechanism run in the original working directory. Since the Markdown package executes a Lua interpreter, which expects to find auxiliary files produced by the package in the current working directory, this presents a problem.

To solve the problem, version 2.5.4 of the Markdown package introduced the new `outputDir` option, which informs the Lua interpreter where it should look for the auxiliary files. Create a text document named `document.tex` with the following content:

```
\documentclass{article}
\usepackage[outputDir=/dev/shm]{markdown}
\begin{document}
\begin{markdown}
A First Level Header
=====
A Second Level Header
-----
Now is the time for all good men to come to the
aid of their country. This is just a paragraph.
\end{markdown}
\end{document}
```

Execute the following command to produce a document with a single section, subsection, and paragraph, redirecting auxiliary files to `/dev/shm` (the RAM disk available on recent Linux kernels):

```
pdflatex -output-directory /dev/shm \
-shell-escape document.tex
```

2.2 The Lua command-line interface

The Markdown package hands markdown documents to a Lua parser. The parser converts them to \TeX and hands them back to the package for typesetting (see Figure 1). This procedure has the advantage of being fully automated. However, it also has several

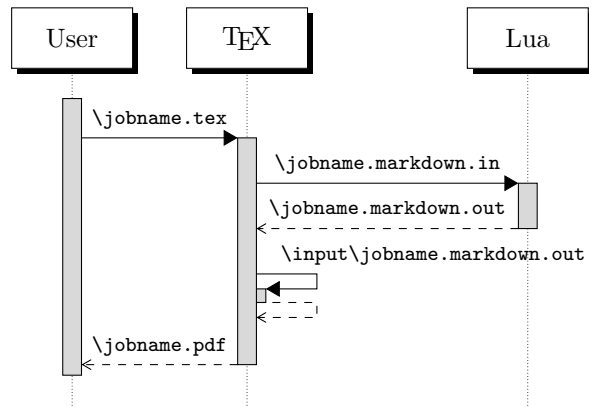


Figure 1: A sequence diagram of the Markdown package typesetting a markdown document using the TeX interface.

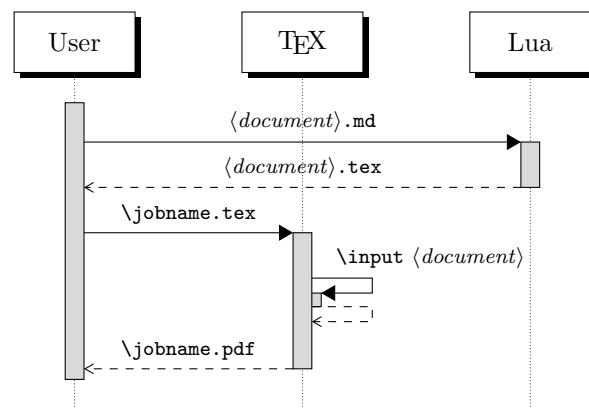


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface.

important disadvantages: The converted TeX documents are cached on the file system, taking up an increasing amount of space. Unless the TeX engine includes a Lua interpreter, the package also requires shell access, which opens the door for a malicious actor to access the system. Last, but not least, the complexity of the procedure also impedes debugging.

A solution to the above problems is to decouple the conversion from the typesetting. First, the user converts markdown documents to TeX. Then, she typesets the TeX documents using the `\input` TeX command (see Figure 2). Before the first step, the user can remove any previously cached TeX documents. Before the second step, she can transform the TeX documents according to her need. During the second step, she does not need to provide shell access to TeX. Since the individual steps are separated, the source of an error is immediately obvious.

To enable this workflow, version 2.6.0 of the Markdown package introduced the Lua CLI, which

is a separate Lua program that can be executed from the shell. Create a text document named `example.md` with the following content:

```
Some of these words are emphasized.
Use two asterisks for strong emphasis.
```

Next, execute the `kpsewhich markdown-cli.lua` command to find the location of the Lua CLI, such as `/usr/local/texlive/2019/texmf-dist/scripts/markdown/markdown-cli.lua` on GNU/Linux with TeX Live 2019. Execute `texlua <location of the Lua CLI> -- example.md example.tex` to convert the `example.md` markdown document to TeX. Finally, create a text document named `document.tex` with the following content:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\input example
\end{document}
```

Execute the `pdflatex document.tex` command to produce a document with one formatted paragraph.

2.3 Documenting L^AT_EX packages

The `doc` L^AT_EX package makes it possible to document a L^AT_EX document by writing a second L^AT_EX document in the comments. This approach to documentation is referred to as literate programming and is popular with L^AT_EX package authors, as it keeps the documentation close to the documented code.

To encourage contributions and readability, documentation can benefit from the use of a lightweight markup language such as Markdown. To allow this use case, version 2.6.0 of the Markdown package introduced the `stripPercentSigns` option, which informs the Lua interpreter that it should strip percent signs from the beginnings of lines in a markdown document. Create a text document named `document.dtx` with the following content:

```
% \iffalse
\documentclass{ltxdoc}
\usepackage[stripPercentSigns]{markdown}
\begin{document}
\DocInput{document.dtx}
\end{document}
% \fi
% \begin{markdown}
% * Candy.
% * Gum.
% * Booze.
% \end{markdown}
```

Execute the following command to produce a document with an unordered list:

```
pdflatex -shell-escape document.dtx
```

2.4 Content slicing

Despite its simplicity (or perhaps because of it), Markdown has become a popular choice for writing all kinds of documents ranging from notes and lecture slides to books that span hundreds of pages. When typesetting these documents, it is often useful to typeset only a small part of them: Lecture slides spanning an entire term can be chopped into lectures. Bits and pieces from a ragtag of notes can be put together into a single coherent document. A behemoth of a book that takes thirty minutes to compile may take only one, when a single chapter is requested.

To make markdown documents more easily stylable, there exist syntax extensions for assigning HTML attributes to markdown elements. To enable typesetting only a part of a markdown document, version 2.7.0 of the Markdown package provides the `headerAttributes` and `slice` options. The `headerAttributes` option enables the Pandoc syntax for HTML attributes and the `slice` option specifies which part of a document should be typeset. Create a text document named `document.tex` with the following content:

```
\documentclass{article}
\usepackage[headerAttributes]{markdown}
\usepackage{filecontents}
\begin{filecontents*}{example.md}
# Palačinky
Crêpe-like pancakes, best served with jam.

## Step 3 {#step3}
Repeat step 2 until no batter is left.

## Step 1 {#step1}
Combine the ingredients and whisk until you
have a smooth batter.

## Step 2 {#step2}
Heat oil on a pan, pour in a tablespoonful of
the batter, and fry until golden brown.
\end{filecontents*}
\begin{document}
\markdownInput[slice=~ ^step3]{example.md}
\markdownInput[slice=step1 step2]{example.md}
\markdownInput[slice=step3]{example.md}
\end{document}
```

Execute the following command to produce a document with one section and three subsections:

```
pdflatex -shell-escape document.tex
```

3 Documentation

Presentation software, Tufte [6] argues, carries its own cognitive style that impedes communication. Similarly, literate programming tends to produce documentation that is poorly structured, because it

adheres too closely to the documented code. Some \LaTeX packages provide a user manual that is written independently of the documented code. While this improves readability, it also sacrifices literate programming and its ease of maintenance.

Before version 2.5.6, the Markdown package only provided technical documentation produced by literate programming. Since version 2.5.6, the package also provided a user manual aimed at the end user rather than a developer. This manual was, however, still produced using literate programming, leading to poor text structure. Since version 2.7.0, the user manual is combined from three texts describing user interfaces, package options, and markdown tokens, respectively. This leads to readable documentation without sacrificing literate programming.

4 Conclusion

\TeX is a fine tool for typesetting many kinds of documents. It may, however, not be the best language for writing them. When preparing structurally simple documents, lightweight markup languages such as Markdown are often the best choice. In this article, I described the new features and documentation in version 2.7.0 of the Markdown package.

Acknowledgments

I gratefully acknowledge the funding received from the Faculty of Informatics at the Masaryk University in Brno for the development of the package.

Support for content slicing was graciously sponsored by David Vins and Omedym.

References

- [1] M. Dominici. An overview of Pandoc. *TUGboat* 35(1):44–50, 2014. tug.org/TUGboat/tb35-1/tb109dominici.pdf.
- [2] J. Gruber. Daring Fireball: Markdown, 2013. daringfireball.net/projects/markdown.
- [3] J. MacFarlane. Pandoc: A universal document converter, 2019. pandoc.org.
- [4] F. Mittelbach. *The doc and shortvrb Packages*, 2018. ctan.org/pkg/doc.
- [5] V. Novotný. Using Markdown inside \TeX documents. *TUGboat* 38(2):214–217, 2014. tug.org/TUGboat/tb38-2/tb119novotny.pdf.
- [6] E. R. Tufte. *The Cognitive Style of PowerPoint*. Graphics Press Cheshire, CT, 2nd edition, 2006.

◇ Vít Novotný
Nad Cihelnou 602
Velešín, 382 32
Czech Republic
[witiko \(at\) mail dot muni dot cz](mailto:witiko@atmail.muni.cz)
<https://github.com/witiko>

New front ends for T_EX Live

Siep Kroonenberg

Abstract

The 2018 release of T_EX Live saw new front ends for T_EX Live Manager and for the installer. This article describes tlshell, which is one of the two new T_EX Live Manager front ends, and the new installer GUI.

There is also a new Java-based front end for T_EX Live Manager by Norbert Preining, tlockpit, but that is not discussed here.

1 Introduction

During 2017 and 2018, a few new GUIs have been built for T_EX Live Manager and for the T_EX Live installer.

The graphical versions of the installer and of T_EX Live Manager date back from the 2008 overhaul of T_EX Live. The Perl/Tk-based GUIs were showing their age, so it was getting to be time for a fresh start, based on more up-to-date technology.

2 Tlshell

The new tlshell GUI for T_EX Live Manager (fig. 1) is a somewhat simpler version of the Perl/Tk GUI mode of T_EX Live Manager. The only option on offer for multiple repositories is a checkbox for adding or removing ‘tcontrib’, which contains packages which cannot be part of T_EX Live for one reason or another.

I hope that tlshell still offers the features that most users need. Anyhow, all features of T_EX Live Manager are still available via the command-line, and tlshell is not confused by multiple repositories configured by other means (fig. 2).

3 Installer

The new installer GUI offers roughly the same configuration options as the old one, the initial basic mode matching the old wizard installer, and the advanced

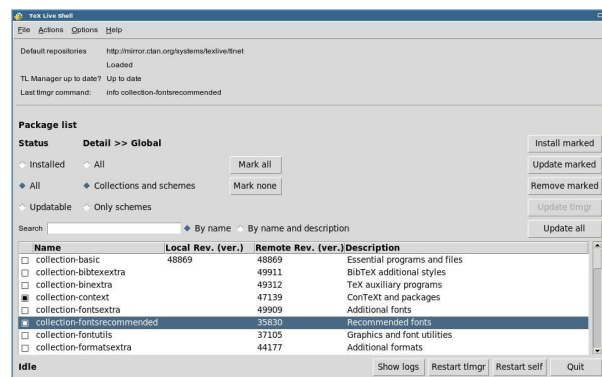


Figure 1: Tlshell, a GUI for T_EX Live Manager

Siep Kroonenberg

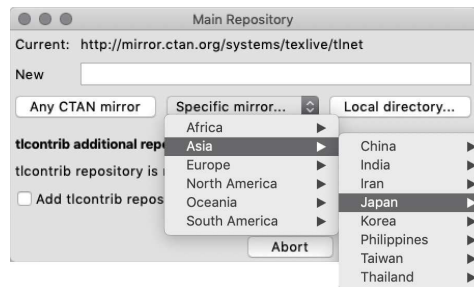


Figure 2: Tlshell configuring repositories (macOS)

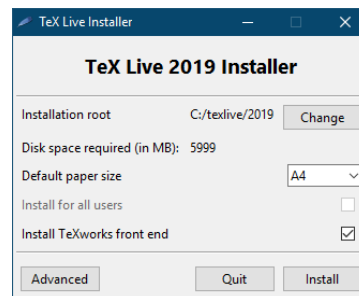


Figure 3: The installer, starting up in basic mode (Windows). Note the ‘Advanced’ button.

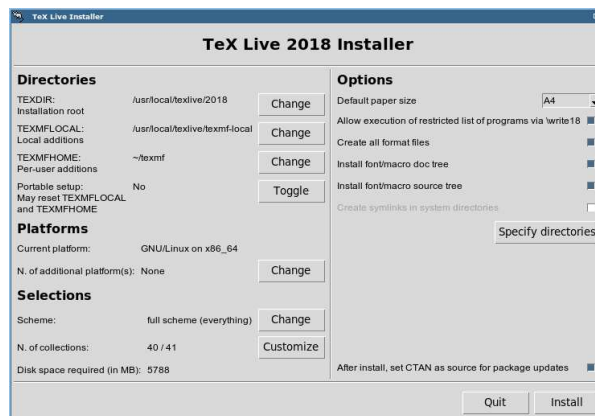


Figure 4: The installer, after pressing the ‘Advanced’ button (GNU/Linux).

mode, triggered by an ‘Advanced’ button, matching the Perl/Tk advanced GUI (figs. 3, 4).

The dialog for selecting the installation root, which was introduced soon after the official 2018 release, has been adapted for the new GUI (fig. 5).

4 Separating front end and back end

For the new GUIs, Norbert and I opted for building GUI front ends as separate programs from a text-mode back end. We were leery of committing ourselves to another Perl extension library which might run out of steam, and hope to have better luck with standalone GUI technology.

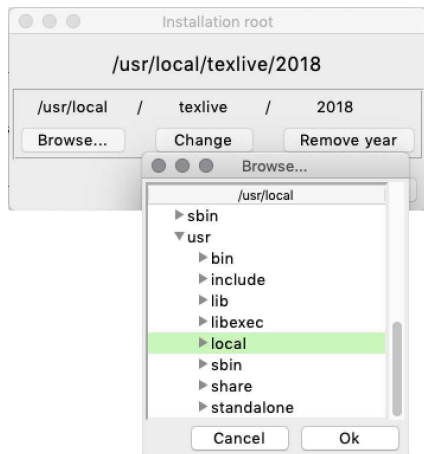


Figure 5: Dialog with directory browser for the installation root (macOS). On Windows the native directory browser is used instead.

4.1 Conversations

If the text mode back end is organized as a conversation, say, consisting of user requests and program responses, then this conversation can be diverted through a graphical front end.

A graphical front end can represent options and commands as buttons, and can display results within the GUI in a variety of ways, e.g. raw text output can go to a scrolling text window, or the output can be interpreted and used to update displayed information.

For \TeX Live Manager, Norbert introduced a new shell mode as such a conversation: users can type their requests within \TeX Live Manager, which typically match normal \TeX Live Manager command-line commands. \TeX Live Manager then carries out those requests, handles the resulting output, and keeps running to receive more requests.

4.2 The installer as a conversation

As to the installer: the text mode installer has been such a conversation from day one. Its text layout mimics the graphical installer, but it is in essence a conversation, with mostly single-letter user commands, and with replies in the form of a rewritten text screen with appropriate changes.

All three old installer menu interfaces load their own Perl include file, which finishes by informing the main installer of the choices made.

A new fourth Perl include file now takes care of communicating with the GUI front end. Only minor changes were needed in the main installer code. In writing this include file I took some cues from the text installer.

4.3 Data

A drawback of a separate front end is that it does not have automatic access to the internals of the back end. The conversation must take care of transmitting data from the back end to the front end and back, not only at the beginning and at the end, but also when the front end wants an update on, for instance, the required disk space. Some duplication between front end and back end is unavoidable, but it was not too much of a problem.

5 Tcl/Tk

I built the new installer GUI and `tlshell` with the Tcl/Tk scripting language. Frankly, I did not do a lot of research into other options. I am familiar with Tcl/Tk and like it: it is simple, small, cross-platform, is well-supported and has good backward compatibility. Plus, I could borrow ideas and solutions from the old Perl/Tk GUIs.¹

As to platform support: Tcl/Tk is already part of macOS, and is packaged for all mainstream GNU/Linux distributions.

For Windows, compiling or cross-compiling Tcl/Tk and adding it to \TeX Live proved to be no particular problem. Even better, thanks to Keene's Kit-Creator project (kitcreator.rkeene.org/fossil), I could build Tcl/Tk as a single-file executable.

6 Localization

The `msgcat` Tcl core package provides localization support. This package tries to determine the system's locale, but it is also possible to set a locale explicitly.

The built-in function for translating strings uses Message Catalogs. Typically, these are generated from `.po` files such as those used by the previous \TeX Live GUIs.

In order to accommodate translators, the Tcl/Tk GUIs use those same `.po` files directly, with help of a tcl version of Norbert Preining's translation function written for \TeX Live. Therefore, any translatable string which was already in the old GUIs will be picked up by the new ones.

◇ Siep Kroonenberg
 siepo (at) bitmuis dot nl
<https://tug.org/texlive>

¹ The Tk in Perl/Tk is a derivative of the Tk from Tcl/Tk, but has not kept up with Tcl/Tk itself. For the 2019 release, Perl/Tk will no longer be present in \TeX Live's built-in distribution of Perl for Windows.

TinyTeX: A lightweight, cross-platform, and easy-to-maintain L^AT_EX distribution based on T_EX Live

Yihui Xie

Abstract

As a L^AT_EX user for 15 years, I have suffered from two problems related to the installation of L^AT_EX and maintenance of packages: 1) The full versions of common L^AT_EX distributions are often too big, whereas the smaller basic versions often lack packages that I frequently use; 2) It is tedious to figure out which missing packages to install by reading the error log from the L^AT_EX compilation. TinyTeX (<https://yihui.name/tinytex/>) is my attempt to address these problems. The basic version of TinyTeX is relatively small (150MB on Linux/macOS when installed), and you only install additional packages if/when you actually need them. Further, if you are an R user, the installation of missing packages can be automatic when you compile L^AT_EX or R Markdown documents through the R package `tinytex`.

1 Motivation

If you do not want to be bothered by L^AT_EX errors that tell you certain class or style files are missing, one way to go is to install the full version of the L^AT_EX distribution, which typically contains the vast majority of packages on CTAN. Take T_EX Live for example. The size of its full version is 4 to 5GB. Yes, I do hear the argument that hard disk storage is fairly cheap today. Why should this 5GB bother us at all? The problems are:

- It can take a long time to download, although we usually do this only once a year. However, if you use a cloud service for continuous integration or testing (e.g., Travis CI) of your software package that depends on L^AT_EX, this can be worse, because each time you update your software (e.g., though a GIT commit), the virtual machine or cloud container downloads 5GB again.
- It contains a lot of L^AT_EX packages that an average user does not need. I do not know if I'm a representative user, but for the more than 5600 packages on CTAN, I routinely use less than 100 of them. In other words, I'm just wasting my disk space with more than 98% of the packages.
- It takes much longer to update packages if you choose to update all via `tlmgr update --all` (and you will be installing newer versions of packages that you do not need, too).

Without installing the full version, you may be confused when compiling a document and a needed

package is not installed. The report at github.com/rstudio/rmarkdown/issues/39 is a good example to show how users can be confused. The main reason for the confusion is that an error message like the one below does not tell you how to resolve the issue (i.e., which package to install and how to install it):

```
! Error: File 'framed.sty' not found.
Type X to quit or <RETURN> to proceed,
or enter new name. (Default extension: sty)
Enter file name:
```

Even worse, T_EX Live can be different on different platforms. For example, if you use a Linux distribution's packaging of T_EX Live, typically you cannot just install the (system) package named `framed` even if you know `framed.sty` is from the (L^AT_EX) package `framed`, because T_EX Live is often made available by distributions as *collections* of (L^A)T_EX packages, so you have to figure out which system package contains the L^AT_EX package `framed`. Is it `texlive-framed`, or `texlive-latex-extra`? On another front, if you use MacT_EX (which is essentially T_EX Live) on macOS, you would usually run `sudo tlmgr install framed`, hence type your password every time you install a package.

Then the next year when a new version of T_EX Live is released, you may have to go through the same pain again: either waste your disk space, or waste your time. One interesting thing I noticed from macOS users was that many of them did not realize that each version of MacT_EX was installed to a different directory. For example, the 2018 version is installed under `/usr/local/texlive/2018`, and the 2017 version is under `/usr/local/texlive/2017`. When they started to try TinyTeX (which recommended that they remove their existing L^AT_EX distribution), they had realized for the first time that there were five full versions of T_EX Live on their computer, and they were very happy to suddenly regain more than 20GB of disk space.

I wished there were a L^AT_EX distribution that contains only packages I actually need, does not require `sudo` to install packages, and is not controlled by system package managers like `apt` or `yum`. I wished there were only one way to manage L^AT_EX packages on different platforms. Fortunately, the answer is still T_EX Live, just with a few tricks.

2 The infraonly scheme and the portable mode to the rescue!

There are three possible ways to cut down the size of T_EX Live:

1. Only install the packages you need.
2. Do not install the package source.
3. Do not install the package documentation.

The first way can be achieved by installing a minimal scheme of T_EX Live first, which includes its package manager `tlmgr`, and then install other packages via `tlmgr install`. The minimal scheme is named `scheme-infraonly`, and it is only about 10MB.

The second and third ways can be specified through installation options, which I will mention soon. The package documentation contributes a considerable amount to the total size of a T_EX Live installation. However, I have to admit I rarely read them, and I do not even know where these documentation files are on my computer. When I have a question, I will almost surely end up in a certain post on tex.stackexchange.com, and find a solution there. It is even rarer for me to read the package source files, since I am not a L^AT_EX expert, nor am I interested in becoming an expert.

With the network installer of T_EX Live (tug.org/texlive/acquire-netinstall.html), we can put the above pieces together, and automate the installation through an “installation profile” file. Below is the one that I used for TinyTeX (named `tinytex.profile`):

```
selected_scheme scheme-infraonly
TEXDIR ./
TEXMFSYSCONFIG ./texmf-config
TEXMFLOCAL ./texmf-local
TEXMFSYSVAR ./texmf-var
option_doc 0
option_src 0
option_autobackup 0
portable 1
```

The installation is done through

```
./install-tl -profile=tinytex.profile
```

where `install-tl` is extracted from the Net installer (use `install-tl-windows.bat` on Windows). The full source of the installation scripts can be found on Github at github.com/yihui/tinytex/tree/master/tools. To install TinyTeX on *nix, run `install-unx.sh`; to install it on Windows, run `install-windows.bat`.

I set the `portable` option to 1 above, which means the installation directory will be portable. You can move it anywhere in your system, as long as you know how to handle the `PATH` variable, or call the executables (e.g., `tlmgr` or `pdflatex`) with their full paths. By default, the installation scripts of TinyTeX will try to add T_EX Live’s bin path to the environment variable `PATH`, or create symlinks to a path that is in `PATH` (e.g., `/usr/local/bin` on macOS and `$HOME/bin` on Linux).

A portable installation without admin privileges also means anyone can install and use T_EX Live on

any platforms supported by T_EX Live. You can also install a copy to a USB device and use it from there. Users inside an institute no longer need to ask for IT help with managing L^AT_EX packages because of the powerful and useful `tlmgr`. With TinyTeX, `tlmgr` is the one and only way to manage packages directly, and you will not need `sudo`, `apt`, or `yum`.

3 The R package `tinytex`: install missing L^AT_EX packages on-the-fly

Now I only have one last wish for T_EX Live: I wish it could install missing packages on-the-fly like MiK_TE_X when compiling documents. I do not know how MiK_TE_X implemented it. I’m primarily an R [2] package developer. I do not know much about the T_EX language or Perl. I know how to search for the package that contains a certain style or class file and install it, e.g.,

```
$ tlmgr search --global --file "/times.sty"
psnfss:
    texmf-dist/tex/latex/psnfss/times.sty
...
$ tlmgr install psnfss
```

I had done this too many times in the past, and thought it might be possible to automate it. I made an attempt in the R package `tinytex` [3]. I guess L^AT_EX experts may frown upon my implementation, but it was the best I could do, given my limited capabilities and knowledge in L^AT_EX.

Basically, I try to compile a L^AT_EX document via an engine like `pdflatex` or `xelatex`, with arguments `-halt-on-error` and `-interaction=batchmode`. If the exit status is non-zero, I will parse the error log and find the error messages. If I made any contribution at all, it would be the following possible error messages that I collected in about a year:

```
! \LaTeX{} Error: File ‘framed.sty’ not found.
/usr/local/bin/mktexpk: line 123: mf:
  command not found
! Font U/psy/m/n/10=psyr at 10.0pt not
loadable: Metric (TFM) file not found
!pdfTeX error: /usr/local/bin/pdflatex
(file tcrm0700): Font tcrm0700 at 600 not found
! The font "FandolSong-Regular" cannot be found.
! Package babel Error: Unknown option
‘ngerman’. Either you misspelled it
(babel)                or the language
  definition file ngerman.ldf was not found.
!pdfTeX error: pdflatex (file 8r.enc):
  cannot open encoding file for reading
! CTeX fontset ‘fandol’ is unavailable in
  current mode
Package widetext error: Install the
  flushend package which is a part of sttools
Package biblatex Info: ... file
```

```
'trad-abbrev.bbx' not found
! Package pdftex.def Error: File
'logo-mdpi-eps-converted-to.pdf' not found
```

In the R package `tinytex`, I try to obtain the names of the missing files or fonts or commands (e.g., `framed.sty`, `mf`, `tcrm0700`), run `tlmgr search` to obtain the package name, and `tlmgr install` the package if possible.

The thing that T_EX Live experts may frown upon is that since I do not know all possible missing packages beforehand, I will just keep trying to compile the document, find the missing packages, and install them. In other words, I do not know if there is a missing package unless I actually compile the document and hit an error. If a document contains n missing packages, it may be recompiled n times.

On the bright side, this only needs to be done at most once for a document, so even if it is slow for the first time, the compilation will be much faster next time because all necessary packages have been installed. The process is also automatic (by default), so all you need to do is wait for a short moment. This feature is turned on for R Markdown [1] users, which means if the user's L^AT_EX distribution is TinyTeX, they will almost never run into the issue of missing packages when compiling R Markdown to PDF, and the “easy-to-maintain” TinyTeX should not need maintenance at all. As a matter of fact, this article was written in R Markdown, and the first time I compiled it, the `tugboat` package was automatically installed:

```
tlmgr search --file --global /ltugboat.cls
tlmgr install tugboat
...
[1/1, ??:??/?:??:?] install: tugboat [26k]
running mktexlsr ...
done running mktexlsr.
```

The other major thing `tinytex` does is to emulate `latexmk`, i.e., try to compile a L^AT_EX document till all cross-references are resolved. The reason to reinvent `latexmk` in an R package is that `latexmk` cannot install missing packages on-the-fly.

To sum it up, if R users compile a L^AT_EX document via `tinytex`, usually they will not need to know how many times they need to recompile it, or run into errors due to missing packages. My implementation may be clumsy, but the reaction from users seems to be positive anyway: github.com/yihui/tinytex/issues/7. I hope this could give some inspiration to developers in other communities, and I will be even more excited if T_EX Live adds the native (and professional) support someday, so I can drop my poor implementation.

Yihui Xie

4 Discussion

There is no free lunch. TinyTeX also has its drawbacks, and you have to consider whether they matter to you. First of all, when installing TinyTeX, you are always installing the very latest version of T_EX Live. However, as I have mentioned, TinyTeX is a portable folder, so you can save a copy of a certain version that worked for you, and use it in the future.

Secondly, the installation of TinyTeX and the (automatic) installation of additional L^AT_EX packages requires an Internet connection. This may be the biggest drawback of TinyTeX. If you plan to work offline, you will have to make sure all packages have been installed in advance.

Thirdly, TinyTeX was created mainly for individual users who install TinyTeX for themselves. If a sysadmin wants to install a shared copy of TinyTeX for multiple users, there will be more technical details to learn (in particular, issues related to permissions, the “user mode”, and packages that are not “relocatable”). I have mentioned them on the FAQ page: yihui.name/tinytex/faq/.

Lastly, TinyTeX is essentially a version of T_EX Live installed through an installation script. I did not provide prebuilt binaries, even though it would be easy technically. I do not fully understand the T_EX Live license and L^AT_EX package licenses, but I guess I would be very likely to violate these licenses if I provide binaries without also shipping the source files inside at the same time. Anyway, installing TinyTeX over the Internet usually takes only a minute or two, so this may not be a big concern.

I hope you might find TinyTeX (and the R package `tinytex`, if you happen to be an R user, too) useful. If you have any feedback or questions or bug reports, please feel free to post them to the Github repository: github.com/yihui/tinytex.

References

- [1] J. Allaire, Y. Xie, et al. *rmarkdown: Dynamic Documents for R*, 2019. R package version 1.12.2. rmarkdown.rstudio.com
- [2] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. www.R-project.org
- [3] Y. Xie. *tinytex: Helper Functions to Install and Maintain T_EX Live, and Compile L^AT_EX Documents*, 2019. R package version 0.11.2. github.com/yihui/tinytex

◇ Yihui Xie
RStudio, Inc.
xie (at) yihui dot name
<https://yihui.name>

Extending primitive coverage across engines

Joseph Wright

1 The pdfTeX situation . . .

In recent years, development of pdfTeX has intentionally been limited, with the v1.40 branch now being around for over 10 years. However, in the past there were plans for a v1.50 branch, and some code was written. One primitive that was fully coded-up at that time was `\expanded`. The idea of this is pretty simple: it carries out full expansion like `\message` (and *almost* like `\edef`), but is itself expandable. This highly useful idea made it into LuaTeX (which was initially based on the pdfTeX development code), but until recently wasn't in released pdfTeX itself.

2 . . . vs. the XeTeX situation

XeTeX was primarily written to extend ε -TeX with full Unicode support, as well as loading system fonts. Its development started from ε -TeX, rather than from pdfTeX, which had added various new primitives on top of ε -TeX. Many of pdfTeX's additions to ε -TeX have to do with directly producing PDF output (ε -TeX supports only DVI), but others are entirely independent of that.

Over the years, some of these other “utilities” have been added to XeTeX (for example `\pdfstrcmp`, which in XeTeX is just `\strcmp`). However, several have not made it, but *have* been added to pTeX and upTeX. That has meant that XeTeX has been “a bit behind” in feature terms: some things simply can't be done without primitive support.

3 A development opportunity arises

Recently, a Travis-CI testing environment has been created for TeX Live (see github.com/TeX-Live/texlive-source), meaning that it's now *easy* to try adding new material to the WEB sources of pdfTeX, XeTeX, etc. As part of more general work on primitives, it made sense to bring XeTeX “back in line” with (u)pTeX. That's important for expl3, as the L^ATeX team have been using almost all of the primitives that were “missing” in XeTeX, as well wanting to bring `\expanded` into the mainstream.

4 Providing `\expanded`

For some time, the L^ATeX team have been thinking about asking for `\expanded` to be made more widely available. Unlike the `\romannumeral` “trick”, `\expanded` does not require any hard work to get “past” any output, so it is very useful for creating macros that work like functions. It's also fast and clear in intention.

The code itself was easy enough to move around: a bit of copy-pasting! As well as merging into the stable branch of pdfTeX, I worked out how to add `\expanded` to XeTeX and the Japanese TeX engines pTeX and upTeX. So soon we'll all be able to do

```
\def\{a\}\{b\}\{c\}
\message{Hello \a\space #}
\detokenize\expandafter
  {\expanded{Hello \a\space #}}
\bye
```

(Try the example in LuaTeX if you don't have the burning edge pdfTeX binaries.)

5 New primitives in XeTeX

So, besides `\expanded`, what has been added? The new additions are all named without the pdf prefix that pdfTeX includes, as they have nothing to do with PDFs (and XeTeX is not pdfTeX):

```
\creationdate \elapsedtime \filedump
\filemoddate \filesize \resettimer
\normaldeviate \uniformdeviate \randomseed
```

These enable things like random numbers in the L^ATeX3 FPU, measuring code performance, and checking the details of files: all stuff that is in expl3 will now work with XeTeX.

I should add that although I did the grind of working out how to integrate the pdfTeX code into XeTeX, Akira Kakuto sorted out the areas that needed knowledge of C, in particular where XeTeX's Unicode internals don't match up with pdfTeX's 8-bit ones.

6 Adjusting `\Ucharcat`

I made one other minor adjustment to XeTeX: altering how `\Ucharcat` works so it can create category code 13 (“active”) tokens. That probably won't show up for users; however, it helps the team extend some low-level expl3 code. It should just mean one fewer XeTeX restriction.

7 Getting the code

TeX Live gets binary updates only once per year, so users there will need to wait for the 2019 release. On the other hand, MiKTeX already has the new features, so if you are on Windows it's pretty trivial to try. If you use TeX Live and want to test this out, you can update your binaries in-place, for example from W32TeX (w32tex.org): if you understand what that means, you probably know how to do it!

◇ Joseph Wright
Northampton, United Kingdom
joseph dot wright (at)
morningstar2.co.uk

ConTeXt LMTX

Hans Hagen

1 Introduction

More than decade after introducing the ConTeXt version for LuaTeX, flagged MkIV (the version known as MkII ran under more traditional TeX engines), it is time for something new. As MkVI, MkIX and MkXI are already taken for special variants of MkIV files a new acronym was coined: LMTX. Such a sequence of letters must sound somewhat fancy but it actually has a meaning, or better: it has several. One is that it stands for LuaMetaTeX, an indication of an engine to be used. But one can also think of the components that make up ConTeXt code: Lua, MetaPost, TeX, and XML. But once it is clear to a user what (s)he is dealing with, it can also indicate a “Lean and Mean TeX eXperience”. But, as with MkIV, eventually it will become just a tag.

So, with this out of the way, the next question is, what do we mean by LuaMetaTeX? The term MetaTeX first surfaced at a ConTeXt meeting but it is actually a variant of what Taco and I always thought of when we discussed LuaTeX: an engine that hosts not only TeX but also MetaPost, which we all like quite a lot. But, as Lua is now an integral part of this adventure, it has been glued to more than just the engine name.

In the next sections it will be clear what we’re actually talking about and how this relates to ConTeXt LMTX. But let me first stress that after a decade of usage, I’m convinced that the three main components fit nicely together: TeX has a good track record of usability and stability; MetaPost is a nice graphical description language, efficient and very precise, and Lua is, in my opinion, the nicest scripting language around, minimal, rooted in academia and developed in a steady and careful way. It makes for a hard to beat combination of languages and functionality.

2 Using ConTeXt

The average ConTeXt user (MkIV) needs only the LuaTeX engine, a bunch of macros, fonts and hyphenation patterns. One can install ConTeXt from the TeX Live distribution (or via a distribution’s software package) or directly from the ConTeXt garden. The first ships the yearly, so-called ‘current’ release; the second also provides intermediate ‘beta’ releases. In both cases the number of installed bytes is reasonable, especially when one compares it to many programs or packages. In fact, relative to many programs that come with a computer ecosystem these

days, a TeX installation is no longer that large.

Most users will use ConTeXt from within a text editor, normally by hitting a key to process; others might run it on the command line. Then there are those who run it as part of a complex workflow where no one sees it being run at all. When you run LuaTeX with ConTeXt, there is the usual log data flying to a console as well as some short delay time involved to get the result. But that is the game so there is nothing to worry about: edit, run, wait and watch.

On a multi-core machine, only one CPU core will be taken and, depending on the document, a reasonable amount of memory. As ConTeXt evolves we try to limit its use of resources. If we take almost any browser as benchmark then TeX is cheap: its memory consumption doesn’t slowly grow to persistent gigabytes, there is no excessive (unnoticed but not neglectable) writing to disk, and when one is just editing a document it will not suddenly take CPU cycles. We try not to turn the ConTeXt+LuaTeX combination into bloatware.

If you listen to what is discussed between the lines at the ConTeXt meeting you will notice that some use this rendering system as a sort of appliance: it does the job without the end user of the document ever realizing what is involved. Using some virtual machine in this case is quite normal. When ConTeXt is running on a server or background system we need to keep in mind that performance is not required to improve that much and that low power solutions are considered more often. This also means that we must try to lower the memory footprint as well as the need for CPU cycles.

There are many cases where the system is used to generate (few) page documents as part of a larger workflow (or program). One example is creating a PDF file from a MetaPost graphic (with TeX) that gets converted to SVG and then ends up in a web page (part of a status/input screen). It’s hard to beat MetaPost in terms of quality and cost effectiveness. For this a lean and mean and well contained setup is needed. But, in order to be permitted to use such a subsystem one could be asked to prove that what’s underneath is not some complex, hard to maintain component. It being open source is not enough.

Users find it hard to convince employers to use a TeX system. It is seen as large, is considered old, doesn’t always fit in. Therefore, contrary to what one expects, expensive, less robust and less future safe solutions are chosen. How we can help users addressing this problem was also a topic discussed at the last ConTeXt meeting.

All these aspects (of usage) have led to what I present now as ConTeXt LMTX, which started mid-

2018 and is expected to be stable mid-2019, after a year of intense experimenting and development. This is in time for the 2019 ConTeXt meeting where we can pick up discussions about how to make sure TeX is a valid candidate for rendering (so far as that is still needed in the browser dominated arena).

3 Packaging ConTeXt

We have now arrived at a brief summary of what ConTeXt LMTX actually is, but let's stress that for the average user it is what they already have: ConTeXt MkIV using LuaTeX. In fact, we will continue to ship what has been there for years alongside LMTX so that users can test and we can fix bugs. Some parts of the code base already have LMTX specific branches but so far (most) users never enter these.

- We use a lean and mean engine: LuaMetaTeX identifying itself as LuaTeX 2.0 (we count from there). The binary is some mere 3 MB which is much smaller than stock LuaTeX.¹ The size matters because the engine is also a Lua processor and used as such.

- The LuaMetaTeX code base is also relatively small and is now part of the ConTeXt distribution. This means that when one downloads the archive or installs ConTeXt, the source is included! One gets the whole package. The 12 MB source tree compresses to around 2 MB.²

- If needed the user can compile the program. The build is relatively simple with essentially no dependencies, very few requirements, and all files that are needed are included. The average user will not compile but it adds to the idea that the user is independent and that, when ConTeXt is used as a component, all batteries are included.³ With the sources in the distribution, users on non-standard systems can easily bootstrap a working system.

- Where LuaTeX depends on a few external libraries, LuaMetaTeX goes with the absolute minimum as it is what the name says: a core TeX engine, the MetaPost machinery, and Lua.⁴ The few libraries that we ship are part of the source tree and their interfaces are unlikely to change.

- There is just one program: LuaMetaTeX. We use Lua 5.4 and no longer need LuaJIT as it lags behind and has no real significant performance ad-

vantages. There are no extra binaries needed, as this one program also serves as a stub. The first experiences have demonstrated that Lua 5.4 is somewhat faster than its predecessors. We plan to use the more efficient generational garbage collector once it becomes stable.

- When it comes to installing ConTeXt, the engine is also the installer. Instead of using `rsync` we use `http`. An initial install can take a little time, but updates much less. Once the installer is unzipped there are no dependencies on any other programs. The small size of the binary facilitates such usage.

- The installation only has the files needed for MkIV. Of course there is still the archive with everything, but there is no need to install MkII files and resources when only LuaMetaTeX is used. At some point the installer will allow the installation of both MkII and MkIV.

- The MkIV codebase is aware of the engine and triggers LMTX mode accordingly. It will use (a few) different files when it runs on top of LuaTeX or LuaMetaTeX. There might be a point in time when we make a more rigorous split.

4 Fundamental changes

A user now thinking “So what?” deserves some more explanation about what has been changed under the hood. Part of the answer relates to the shrunken binary. How that happened is discussed in a document that kept track of the process and decisions but probably is only of interest for a few. A short summary follows.

At some point last year I realized that I was coming up with solutions where LuaTeX was actually working a bit against me. One reason for that is that we had to become stable at some point and could not change it fundamentally any longer. The main reason for that was that other macro packages are also using it: even a trivial change (say in the log) can spam your mailbox with complaints. At the same time it started annoying me that we had to carry around quite a bit of code that ConTeXt doesn't need or use at all. In addition to that, there were some dependencies on foreign code that resulted in occasional (enforced) updates of the rather complex source tree and build structure. The switch to a new PDF introspection library had demonstrated that dependencies could simply be made much less troublesome. But, as TeX is basically just juggling bytes I wondered if it could be done even better (hint: in Lua).

Already a while ago ConTeXt started using its own OpenType font loader, written in Lua instead of the one derived from FontForge. But actually that built-in loader code was the last to go as bits and

¹ Of course we still pay attention to LuaTeX, but there we have more or less frozen the feature set in order to make it a future-safe engine.

² This will be at the formal release at the 2019 meeting.

³ For now, I directly handle the Windows and Linux binaries and Alan Braslau takes care of the OSX and FreeBSD binaries. We have decided on providing 64 bit binaries on these systems that we actively use and in the future they will be generated on the compile farm.

⁴ The only shared libraries that are referenced are `libc`, `libm`, and `libdl`.

pieces were still hooked into the different parts of the code base, for instance in the PDF backend.⁵

So, the first thing to get rid of was image inclusion library code. Much of the inclusion was already under strict ConTeXt control anyway. Of course we still can include images but ConTeXt does it entirely under Lua.⁶

Next to go was font embedding. In ConTeXt we already did most of that in Lua, so this kick-out was not that dramatic. As a natural followup, generating the page stream was also delegated to Lua. For the record: in all these stages I had a hybrid code base, meaning that I could continue to do the same in LuaTeX. Only at some point the diversion was too large for comfort, and I switched to dedicated code for each engine. Also, the regular ConTeXt code had to keep working so every (even temporary) change had to be done very carefully.

Once the built-in backend was reduced that way, I decided to ship out the whole page via Lua. Keep in mind that in ConTeXt MkIV we always only needed part of the backend: we never depended on any of the extensions supported by the engine and did most PDF-specific features differently already. But for macros using these extensions, it's one of the components that for sure will suffer in performance from being written in Lua instead of C code.

We now no longer had any need for the code in the built-in font loader: we didn't use the loading code but the backend still had used some of its magic for inclusion. So, out went the loader. When I realized that TeX needs only a little information to support what we call base mode, I decided that we could also replace the TFM loader by a Lua loader. We already had to deal with virtual fonts in the new backend code anyway. So, basically most font related code is now gone: only a little is passed to the engine now, that which is needed to do the typesetting.

With the backend reduced to a minimum, it was then a small step to removing it altogether. I didn't feel too guilty about this because officially TeX has no backend: it's an extension. So, the `img` and `pdf` libraries are also gone. With the whole PDF file now generated directly by ConTeXt it was time to deal with extensions. Stepwise these were removed too. In the end we only had a generic so-called 'whatsit' left. The impact on the ConTeXt code is actually not that large: the biggest complication is that we need

⁵ Already some time ago I have made sure that this part is not ever used by ConTeXt, which made me confident that at some point the entire library could be removed.

⁶ We still keep the PDF introspection library available as it has some advantages over using Lua, but I have developed a somewhat limited Lua alternative to play with; maybe a possible future variant.

to support both stock LuaTeX and LuaMetaTeX, so figuring out a way to do that (stepwise as we had a transition) took a while. When doing something like this one should not be afraid to write temporary code and later take it out.⁷

I used these opportunities to improve the readability of the uncompressed PDF output a bit (some was already done in LuaTeX). It must be noted that the final virtual font handling happens in the backend (TeX only cares about dimensions) so this can open up a whole new world of possibilities. In principle the produced code can be made a bit more efficient. I must admit that I always treated much of the font code in the backend as a black box. Reverse engineering such code is not my forte (and no fun either) so I tend to just figure it out myself using the formal specifications. That is also more fun.

I didn't mention yet that parallel to this process some more cleanup happened. All code files were touched; where possible, header files were introduced. And every (even small) step was tested by processing documents. In some places the code could be made more efficient because we had less interwoven code as a side effect of removing the backend font and image related code. I also cleaned up a bit of the Lua interface and more might happen there. Some libraries lost features due to the conceptual changes. Also, libraries like `slunicode` were removed as we never truly needed them. In ConTeXt we already used adapted and optimized socket interfaces so we also no longer preload the old ones that come with the socket libraries. There are lots of such details, which is why it took many months to reach this state.

There are fewer command line options and the startup is somewhat streamlined. As in ConTeXt we already are (mostly) `kpse` compatible but entirely in Lua, as that library was removed too. This affected quite a bit of code, however, because the backend is outsourced, also a lot of file handling! Basically only TeX and Lua files are now seen by the frontend. And dealing with system calls was already done in Lua. We don't need much on top of what we accumulated in Lua for over a decade.

One can wonder if we're still talking TeX and the answer (at least for me) is yes we are. Original TeX has only a DVI backend and DVI is nothing more than a simple page description (I can cook up one if I need to). One needs an external program to produce something useful from DVI. Sure, pdfTeX grew to add a PDF backend but that again is an extension, and although LuaTeX separates the code better than

⁷ At some point I will add a DVI backend, just for good old times.

its ancestor, initializations for instance still mess up the code. The only place where extensions and built-in standard functionality, reflected in primitives, overlap is in writing to files. Clearly we do need to support that. However, along with some other (Lua \TeX) primitives, the backend-related ones are gone. But ... we can simply implement them using Lua so that a macro package still sees these primitives. Nowhere is it mandated that a ‘primitive’ should be hardcoded in the engine.

In fact, one reason for going this route is that it is a way to come closer to the original, even as we have a few more primitives (ε - \TeX as well as Lua \TeX). But what about directions, those coming from Aleph (Omega)? It is a fact that in spite of attempts to deal with all these directions, only a few made sense, and Lua \TeX ended up with only four. Of these four, only left-to-right and right-to-left ever worked well. I cannot imagine someone using the vertical ones as they are hard to control. Therefore, as soon as the backend was gone, I decided to keep only the two horizontal directions. Yet, in order to still support vertical rendering boxes got official offsets and orientations, at the \TeX level. Of course the backend is free to interpret this. This might find its way back to stock Lua \TeX but only after I’m satisfied for some time. So, more about this in another article. I kept only the new numeric direction specifiers.

With all this done, simplifying the binary build was on the agenda. This was not trivial. In retrospect I should just have started new, but because of all those dependencies it made more sense to step-wise strip the process to get an idea of what was happening and why. In the end, when it was determined to be sort of impossible to go much smaller, I decided to quit that and just write a bunch of CMake files. The retrospect relates to the fact that this took me a day to figure out for Windows, Linux, OSX and ARM. A nice side effect was that the LuaMeta \TeX engine compiles in about 30 seconds on my about eight-year-old laptop, which suddenly got an extended lifetime (and about 15 seconds for Alan on an almost four-year-old Macbook with an SSD).

5 The roadmap

The roadmap is as follows. Core development took place in Fall 2018 and Spring 2019. On April 1 there was a version (2.00) suitable for use so users could start playing with this variant. At that time, a prototype of the new installer was also ready and tested by some other developers. After this first release I can start optimizing components in Con \TeX t that are currently sort-of hybrid due to the fact that

code has to run on both engines, but that code was not yet distributed. Around the Con \TeX t meeting in Fall 2019 documentation should be available at which time the LuaMeta \TeX source code will also become part of the distribution. Hopefully binaries will then be generated using the compile farm. After that the Con \TeX t code base will become more and more tuned for LMTX. This all has to happen in such a way that no matter what engine is used, Con \TeX t works as expected. In the process we need to make sure that users who (for some reason) use obsolete low level libraries are served by proper replacements.

6 Summary

So what we have now is a lean and mean engine. What are the consequences for users? They get a smaller but all-inclusive package. The lack of single backend and dependencies is also more future proof. Although a native backend performs slightly better for simple text-only documents, any more complex document is processed as fast or faster, and we can hope to gain a bit more over time. For instance, processing a Lua \TeX manual with current Lua \TeX and LMTX code takes 13.0 seconds, while using the native backend takes 12.6. But LuaMeta \TeX with LMTX currently needs 11.7 seconds, so while we lost some on employing more Lua we gained a bit in the engine. The manual still can be processed a bit faster using LuaJIT \TeX , but for other documents the new setup can actually beat that variant by a wide margin. I will not go into details of how this happens because it is probably rather Con \TeX t specific. At any rate, I always try to make sure that whatever I change deep down in Con \TeX t, performance is not hit.

I’m quite satisfied that we now have a clean code base, fast compilation, a bit more Knuthian (less loaded) engine, and that we can distribute all in one package without an increase in size. Combined with the new installer this is quite a step forward in Con \TeX t development, at least for me.

For me, furthermore, the main effect is that I have more freedom for experimenting and prototyping features that could be fed back into Lua \TeX . I also hope that eventually this machinery is well suited for low power computers (and servers) with limited memory. I will probably report more on all this at another time.

I’d like to thank Alan Braslau for his support in this project, his patient testing, and belief in the future. He also made this article a bit more readable. We love these three languages and have lots of plans!

◇ Hans Hagen
<http://pragma-ade.com>

Bringing world scripts to LuaTeX: The HarfBuzz experiment

Khaled Hosny

1 HarfBuzz

Unicode includes thousands of characters and hundreds of scripts,¹ but inclusion in Unicode is just the start. Proper support for many of them is a much more involved process. (Figure 1 shows a few examples.)

To aid Unicode, there is a need for *smart* fonts; fonts that are not merely collections of glyphs. TeX's TFM fonts are a kind of smart fonts, as they contain rules for making ligatures based on certain contexts; but meeting the needs of the world scripts requires more than ligatures. These needs lead to the development of several font and layout technologies that can fulfil them.

One of these technologies is OpenType,² which is widely supported on major operating systems and applications, making it the de facto standard for Unicode text layout. Others include Apple Advanced Typography³ (AAT) and Graphite.⁴

The text layout process can be seen as several successive steps. One particularly interesting and rather complex step is called *shaping*, which basically involves taking chunks of characters that have some common properties (such as having the same font, same direction, same script, and same language) and converting them into positioned font glyphs. A piece of software that does this is often called a *shaper*.

In OpenType the knowledge needed for proper shaping of a given script is split between the shapers and the fonts; a script-specific shaper has embedded knowledge about a certain script (or group of related scripts), and the fonts provide font-specific data that complements the knowledge embedded in the shaper.

One of the widely used OpenType implementations is HarfBuzz,⁵ which identifies itself as a text shaping library. Although HarfBuzz was initially only an OpenType shaping engine, it now supports AAT and Graphite as well. HarfBuzz is an open source library under active development.

2 LuaTeX

LuaTeX is an extended TeX engine with Lua as an embedded scripting language. LuaTeX also supports

Arabic	عربي	Bengali	কর্কি
Devanagari	कर्किक	Gujarati	કર્કિ
Gurmukhi	ਕਰਕਿ	Kannada	ಕರಕಿ
Malayalam	കരകി	Myanmar	ကရကိ
Oriya	କରକି	Sinhala	කරකි
Tamil	கரகி	Telugu	కరకి

Figure 1: Sample texts from some of the world's scripts.

Figure 2: Aref Ruqaa font as rendered with HarfBuzz integration (above) and luaotfload (below).

Unicode text, among other things. The LuaTeX philosophy is that it provides solutions, not answers, so it does not come with an extended text layout engine, and instead provides hooks to its internals so that its users (or macro packages) can extend it as they see fit.

While this is a worthwhile goal, in practice writing a text layout engine for the Unicode age is a complex and demanding task, and takes many person-years to develop. On top of that, it is a moving target as Unicode keeps adding more scripts (both living and dead) and font technologies keep evolving as the problems at hand become better understood,⁶ and it takes quite some effort to remain on top of this.

This has led to having only one mature and feature-full text layout engine for LuaTeX, written purely in Lua by the ConTeXt team. This engine is made available to L^AT_EX users via the luaotfload package as well. It is a fast and flexible engine, and has many interesting features. But it falls short of supporting all scripts in Unicode. Even for the scripts it supports, some fonts might not work well when they utilize rarely used parts of OpenType that the ConTeXt team might not have had a chance to test (figure 2).

HarfBuzz, on the other hand, is considerably more widely used, tested, and exposed to all sorts of

¹ Unicode 12.0 has a total of 137,929 characters and 150 scripts: unicode.org/versions/Unicode12.0.0

² docs.microsoft.com/en-us/typography/opentype

³ developer.apple.com/fonts/

[TrueType-Reference-Manual/RM06/Chap6AATIntro.html](https://truefont.com/TrueType-Reference-Manual/RM06/Chap6AATIntro.html)

⁴ graphite.sil.org

⁵ harfbuzz.github.io

⁶ For example, OpenType had an initial model for shaping Indic scripts, which was later found to be inadequate and a new model was developed (keeping the old model for backward compatibility). Later, a new, more extensible model, called Universal Shaping Engine, was developed to handle many Indic and non-Indic scripts.

tricky and complex fonts.⁷ It also has a larger team of dedicated developers that have spent many years enhancing and fine-tuning it.⁸

3 Integrating HarfBuzz with LuaTeX

Integrating HarfBuzz with LuaTeX would bring the benefits of HarfBuzz without giving up the capabilities of LuaTeX. There have been several attempts to do this, including the one that is going to be discussed here in some detail.

The basic idea is rather simple: get the text from LuaTeX, shape it with HarfBuzz, and then feed the result back to LuaTeX.

LuaTeX provides hooks, called *callbacks*, that allow modifying its internals and adding code to be executed when LuaTeX is about to do certain tasks.

HarfBuzz provides a C API and there are several ways to call such an API from LuaTeX; each has its pros and cons:

FFI Originally part of LuaJIT, but available now for regular LuaTeX as well, this allows binding C APIs without the need for writing separate bindings in C. However, it requires duplicating the C headers of the library inside the Lua code. Using FFI in LuaTeX requires using the less-safe `--shell-escape` command-line option.

Loadable Lua C modules Written in C, this uses the Lua C API for interacting with the Lua interpreter; it can link to any library with a C API (either dynamically or statically). It can be dynamically loaded at runtime like any Lua module (e.g. using `require`), but it is not as well supported by LuaTeX on all platforms.

Built-in Lua C modules Instead of dynamically loading Lua C modules at runtime, they can be statically linked into the LuaTeX binary, making them work on all platforms. This however, requires either building a new independent engine based on LuaTeX, or convincing the LuaTeX team to include the new module.

Making a loadable Lua C module was chosen for this experiment, utilizing the existing `luaharfbuzz` project⁹ and extending it as needed to expose additional HarfBuzz APIs.

In addition to the `luaharfbuzz` module, additional Lua code is needed to extract input from LuaTeX,

⁷ HarfBuzz is used by Mozilla Firefox, Google Chrome, ChromeOS, GNOME, LibreOffice, Android, some versions of Adobe products and many open source libraries, not to mention XeTeX; in all, it has billions of users.

⁸ HarfBuzz started its life around the year 2000 as the OpenType layout code of the FreeType 1 library, long before it was named HarfBuzz.

⁹ github.com/ufyTeX/luaharfbuzz by Deepak Jois

feed it to HarfBuzz and back to LuaTeX, and do any conversion and processing necessary for both input and output to be in the form that both ends can utilize.

4 Loading fonts

LuaTeX's `define_font` callback allows for overriding the internal handling of the `\font` primitive, which can be used to extend the syntax as well as to load additional font formats beyond what LuaTeX natively supports. Although HarfBuzz is not specifically a font loading library, it provides APIs to get enough information for LuaTeX to use the font.

HarfBuzz's font loading functions support only fonts using the SFNT container format,¹⁰ which basically means it supports OpenType fonts (and by extension TrueType fonts, which are a subset of OpenType). It is possible to support other formats by using the FreeType library¹¹ to load the fonts instead of HarfBuzz's own font loading functions, but for the sake of simplicity and to avoid depending on another library this was not attempted. In practice (outside of the TeX world, that is) all new fonts are essentially SFNT fonts of some sort.

Font data in SFNT containers are organized into different *tables*. Each table serves a specific purpose (or several purposes) and has a tag that identifies it. For example, the `name` table contains various font names, the `cmap` table maps Unicode characters to font glyphs, and so on.

The LuaTeX manual describes the structure that should be returned by this callback. Basically, some information about the font is needed, plus some information about the glyphs in the font.

4.1 Loading font-wide data

Loading most font-wide data (font names, format, etc.) is straightforward since HarfBuzz has APIs that expose such information.

There are two main OpenType font flavours based on what kind of Bézier curves is used to describe glyph shapes in the font; cubic Bézier curves (also called PostScript curves, as these are the kind of curves used in PostScript fonts), and quadratic curves (also called TrueType curves, as these are the kind of curves used in TrueType fonts). The main difference between the two is the glyph shapes table: cubic curves use the CFF table, while quadratics use the `glyf` and `loca` tables.

LuaTeX wants the format to be indicated in the font structure returned by the callback (possibly to decide which glyph shapes table to look for, though

¹⁰ en.wikipedia.org/wiki/SFNT

¹¹ freetype.org



Figure 3: Random Unicode emojis using Noto Color Emoji font which embeds bitmap PNGs instead of outline glyphs. (Grayscaled in the print version.)

that seems redundant as it can easily detect it itself). It is easy to determine the format by checking which tables are present in the font, so that is not an issue. However, there is now a different OpenType flavour that does not include any of these tables and instead uses different tables that embed colored glyph bitmaps (used mainly for colored emoji; a few are shown in figure 3). LuaTeX does not support embedding such fonts in PDF files. To work around this, such fonts are identified during font loading, and during shaping (see below) this is detected, and the PNG bitmaps for font glyphs are extracted and embedded as graphics in the document, avoiding the need for including the font itself in the PDF.

4.2 Loading glyph data

Other than font-wide data, LuaTeX also wants to know some information about the font glyphs. Ideally, such information should be queried only when the glyphs are actually being used, and OpenType tables are carefully structured to allow for fast loading of any needed glyph information on demand without having to parse the whole font (which can be time consuming, especially for large CJK fonts containing tens of thousands of glyphs). However, the way things are structured in LuaTeX requires loading all basic glyph information up front. Thankfully, HarfBuzz’s font loading is fast enough that the slowness implicit in loading all required glyph information is not a critical problem.

Although it is theoretically possible not to load any glyphs initially, but wait until after shaping and update the font with information about glyphs that were actually used, this would be very slow as it would happen thousands of times, requiring recreating the LuaTeX font each time a new glyph is used. Also, in my experiments, sometimes glyphs would fail to show in the final document if they weren’t loaded when the font was initially created.

LuaTeX requires all glyphs in the font to have a corresponding character (the font structure seems to make no distinction between characters and glyphs), but not all glyphs in the font are mapped to Unicode characters (some glyphs are only used after glyph

substitutions, e.g. ligatures and small caps). To work around this, pseudo-Unicode code points are assigned to each glyph; LuaTeX characters are full 32-bit numbers, but Unicode is limited to 21-bit values (no code point larger than this will ever be used by Unicode, for compatibility reasons), so the trick is to use the glyph index in the font and then prefix it by 0x110000 (the maximum possible Unicode code point + 1), thus keeping LuaTeX happy by having a character assigned to each glyph. This way any glyph in the font can be accessed by LuaTeX, while not clashing with any valid code point. The downside of this is that any LuaTeX message that tries to print font characters (like overfull box messages) will show meaningless bytes instead. LuaTeX has callbacks that could be used to potentially fix this.

Some font-wide data like ascent, descent and cap-height do not have corresponding entries in LuaTeX fonts and LuaTeX checks instead for the metrics of hard-coded set of characters to derive this information from. To work around this, and since we don’t provide any entries for real characters, we can create fake entries for these characters using the font-wide data instead of the actual character metrics.

Math fonts seem to be tricky as some of the information LuaTeX requires is not exposed by HarfBuzz in a way that can easily be used at font loading time. For example, HarfBuzz has an API to get the math kerning between a pair of glyphs at given positions, but LuaTeX wants the raw math kerning data from the font to do the calculation itself. Handling this properly would require changes to either HarfBuzz or LuaTeX.

5 Shaping

For shaping there are basically two problems to solve: converting LuaTeX’s text representation into something that can be fed to HarfBuzz, and converting HarfBuzz output to a form that can be given back to LuaTeX.

5.1 Converting LuaTeX nodes to text strings

LuaTeX’s default text layout can be overridden with the `pre_linebreak_filter` and `hpack_filter` callbacks. They are called right before LuaTeX is ready to break lines into a paragraph, which is just the right moment to shape the text.

By the time the callbacks are called, LuaTeX has converted its input into a list of *nodes*. Nodes represent different items of the LuaTeX input. Some represent characters/glyphs, some represent glue, while others represent kerning, etc.; there are also

modes for non-textual material like graphics and PDF literals.

HarfBuzz, on the other hand, takes as input strings of Unicode characters, in the form of UTF-8, UTF-16 or UTF-32 text strings, or an array of numbers representing Unicode code points.

Converting character nodes is straightforward; the characters they represent are inserted into the text string. Glue nodes are converted to SPACE (U+0020), and discretionary hyphenation nodes are converted to SOFT HYPHEN (U+00AD). Any other node is converted to OBJECT REPLACEMENT CHARACTER (U+FFFC), which serves as a placeholder that does not usually interact with other characters during shaping, but its presence helps with later converting the HarfBuzz output to LuaTeX nodes.

Now the text is almost ready to be fed to HarfBuzz, but not quite: first it needs to be “itemized”. HarfBuzz takes as input a contiguous run of characters that use the same font and have the same Unicode script, text direction, and language.

Font itemization is grouping together any contiguous run of character nodes that use the same font, along with any intervening non-character nodes (so that glue nodes, for example, are shaped with the text they belong to).

The same goes for Unicode script itemization, except that this depends on the *Unicode Character Database*,¹² which collects many Unicode character properties, including their scripts (HarfBuzz has an API to access these properties). Some characters don’t have an explicit script property, though. Some characters have the script property *Inherit* and these, as you might guess, inherit the script of the preceding character (they are usually combining marks, like accents). Others have the script property *Common*, and they take the script of the surrounding text (they are usually characters that do not belong to a specific script, like common punctuation characters). Unicode Standard Annex #24 describes a heuristic¹³ to handle common characters which suggests special handling of paired characters (e.g. parentheses) so that matching ones get assigned the same script.

Text direction itemization requires first applying the *Unicode Bidirectional Algorithm*,¹⁴ but this was out of scope for this experiment, so users are expected to mark right-to-left segments of the text manually using LuaTeX’s direction primitives, and the code uses this to determine the direction of the text.



Figure 4: Text using Amiri Quran Colored font which uses colored glyph layers to make a distinction between the consonantal text and the later developments of the Arabic script. Black for the base consonants (they just use the text color), dark red for diacritical dots and vowel marks, golden yellow for *hamzah*, and pale green for non-textual elements like the “circled” *āyah* numbers. (The print version is grayscale.)

5.2 Shaping with HarfBuzz

After feeding the input text to HarfBuzz and getting back output, some post-processing is needed.

Some OpenType flavours contain only bitmaps for glyphs (in the CBDT table¹⁵), not outlines; LuaTeX doesn’t know how to embed such fonts in PDF files. These fonts are detected during font loading, and after shaping, the PNG data of such glyphs is extracted using HarfBuzz, then saved to temporary files and finally embedded as graphics in LuaTeX’s node list (it would be better to skip the temporary files step, but there wasn’t any obvious way to do this in LuaTeX). This way the font can be used with LuaTeX without having to actually embed it in the PDF output.

There are also layered color fonts (see figure 4), where the font contains, in addition to regular outline glyphs, a table (COLR¹⁶) that maps some glyphs to layers (composed of other glyphs) and color indices, and another table (CPAL¹⁷) that specifies the colors for each color index. Since LuaTeX doesn’t keep these tables in the font when embedding it into the PDF file (and even if it did, PDF viewers and other PDF workflows are unlikely to handle them), instead the glyphs are decomposed into layers using the relevant HarfBuzz API and the corresponding colors are added using the regular PDF mechanisms for coloring text. (Color transparency is not handled, though, as it requires support from macro packages to manage PDF resources.)

¹⁵ docs.microsoft.com/en-us/typography/opentype/spec/cbdt

¹⁶ docs.microsoft.com/en-us/typography/opentype/spec/colr

¹⁷ docs.microsoft.com/en-us/typography/opentype/spec/cpal

¹² unicode.org/ucd

¹³ unicode.org/reports/tr24/#Common

¹⁴ unicode.org/reports/tr9

5.3 Converting HarfBuzz glyphs to LuaTeX nodes

HarfBuzz outputs positioned glyphs. Output glyph information includes things such as the glyph index in the font and the index of the character it corresponds to in the input string (called *cluster* by HarfBuzz). Glyph positions tell how a given glyph is positioned relative to the previous one, in both X and Y directions (called *offset* by HarfBuzz), as well as how much the line should advance after this glyph in both directions (called *advance* by HarfBuzz, but unlike offsets, only one direction is active at a time, so for horizontal layout the Y advance will always be zero, and for vertical layout the X advance will be zero).

To feed HarfBuzz output back into LuaTeX, a new node list based on the original needs to be synthesized. Using the HarfBuzz *cluster* of each output glyph to identify the node from the original list that this glyph belongs to, we can re-use it in the new list, thus preserving any LuaTeX attributes and properties of the original node.

Character nodes The original node is turned into a glyph node, using the glyph index + 0x110000 as its character (see font loading section above for explanation). If more than one glyph belongs to this node, each gets copied as needed and inserted into the node list, so that all the glyphs inherit the properties of the original node. If the advance width of the glyph is different from the font width of the glyph, a **kern** node is also inserted (after the glyph for left-to-right text, and before it for right-to-left text).

Glue nodes The advance width of the output glyph is used to set the natural width of the glue. This way fonts can have OpenType rules that change the width of the space (e.g. some fonts use a narrower space for Arabic text than for Latin, some fonts kern the space when followed by certain glyphs, and so on).

Discretionary hyphenation nodes The existing pre-line breaking, post-line breaking and replacement node lists¹⁸ of the original node need to be shaped as well. Special handling is needed when characters around a discretionary hyphen form a ligature; when no line breaking happens at that discretionary hyphen then the ligature needs to be kept intact, but when line breaking does happen the text should be shaped as if a real hyphen had been there from the start.

LuaTeX handles this with a **replacement** node list which contains the nodes that should

office coffee HAVANA

of-
fice
cof-
fee
HA-
VANA

Figure 5: Ligatures and kerning are formed correctly around discretionary hyphens, when no line breaking happens, and correctly broken at line breaks.

appear if no line breaking happens, and a **pre** node list that contain what comes before a line break, and **post** for what comes after it. Since ligatures can't just be cut into parts, the text needs to be shaped two times: once with the whole text without a hyphen, and once with the text split into two parts and a hyphen inserted at the end of the first part. It would be very inefficient to reshape the whole paragraph in this manner, and it would also be impractical to store full paragraphs in **replacement**, **pre**, and **post** node lists.

One solution is to reshape just the ligature, but sometimes the shaping output can be different based on the surrounding characters, so cutting the ligature out and shaping it all by itself can produce the wrong result. Fortunately, HarfBuzz has a flag attached to output glyphs that says whether breaking the text before this glyph and shaping each part separately would give the same output or not. We use this flag to find the smallest part of the text that is safe to reshape separately from the rest of the paragraph, starting from the discretionary hyphen, and re-shape only that part. (See figure 5.)

Characters that are not supported by the font are ignored by TeX (no output is shown in the typeset document), and by default only a message is printed in the log file. This is a bit unfortunate as it can be easily missed. With HarfBuzz, unsupported characters return glyph index zero (often named as the **.notdef** glyph), which is usually a box glyph and sometimes has an X inside it to mark unsupported characters. The code will thus insert this glyph into the node list, and since this will effectively disable the missing character messages that LuaTeX outputs, the code emulates LuaTeX behaviour and outputs such messages itself. One side effect of using glyph zero is that even though the character is not shown, the text is preserved in the PDF file and can be searched or copied.

¹⁸ See LuaTeX manual for detailed explanation of these.

5.4 Handling text extraction from PDF

To extract text from a PDF file (e.g. copying or searching), the PDF viewer needs to know how to reverse map glyphs back to Unicode characters. The simplest way to do this is to set the mapping in the font's `/ToUnicode` dictionary, which can handle one-to-one and one-to-many glyph to character mappings (i.e. simple glyphs and ligatures).

Getting one-to-one glyph to character mappings can be partially done at font loading time by reversing the font's `cmap` table. This, however, covers only glyphs that are mapped directly from Unicode characters. In OpenType, not all glyphs are mapped this way, for example, small cap glyphs are not mapped directly from Unicode characters as they are only activated when a certain font feature is on (the characters are first mapped to regular lowercase glyphs, then a *small caps* feature maps those to small cap glyphs), and detecting what characters they came from can happen only after shaping.¹⁹ Because of this, there is still a need to modify the fonts after shaping each part of the text, to update the `ToUnicode` values for each glyph, and for large documents this is rather slow.

Furthermore, `/ToUnicode` can't handle all cases. With HarfBuzz there can be glyph-to-character relationships that are any of one-to-one, one-to-many, many-to-one and many-to-many. With `/ToUnicode` the first two can be handled, but the last two can't. Also, the `/ToUnicode` mapping is required to be unique for each glyph; the same glyph can't be used for different Unicode characters, but that is a possibility in OpenType and other modern font formats.

Fortunately there is another, more general, mechanism in PDF; the `/ActualText` spans, which can enclose any number of glyphs and represent any number of characters (not all PDF viewers support them, though, but we can't help that).

After shaping, HarfBuzz *clusters* are used to group glyphs that belong to one or more characters and that information is stored in the node list. Then, after line breaking, `/ActualText` spans are added for any group that can't be represented in the `/ToUnicode` dictionary. This is done after line breaking (in the `post_linebreak_filter` callback) since there are many restrictions on the kind of nodes that can appear in the node lists of discretionary hyphenation nodes.

¹⁹ The alternative would be to decode the font features and parse them, which requires a substantial effort and would still not handle all cases since features can do different things for different scripts and languages, and there might be more than one way to arrive at the same glyph.

6 Conclusion

Integrating HarfBuzz with LuaTeX is possible and can bring many benefits to LuaTeX and enable more users to enjoy its capabilities. There are some technical issues to solve and rough edges to round, but nothing that would substantially prevent such integration.

The code described here was made possible thanks to generous support from the TUG development fund (tug.org/tc/devfund). The code and the required `luaharfbuzz` module are available at:

github.com/khaledhosny/harf
github.com/ufyTeX/luaharfbuzz

◇ Khaled Hosny
github.com/khaledhosny

L^AT_EX News

Issue 29, December 2018

Contents

Introduction	1
Bug reports for core L^AT_EX 2_ε and packages	1
Changes to the L^AT_EX kernel	1
UTF-8: updates to the default input encoding . . .	1
Fixed <code>\verb*</code> and friends in X _Y L ^A T _E X and LuaL ^A T _E X	1
Error message corrected	2
Fixed fatal link error with <code>hyperref</code>	2
Avoid page breaks caused by invisible commands	2
Prevent spurious spaces when reading table of contents data	2
Prevent protrusion in table of contents lines . .	2
Start L-R mode for <code>\thinspace</code> and friends . .	2
Guarding <code>\pfill</code> in <code>doc</code>	2
Changes to packages in the tools category	3
Sometimes the <code>trace</code> package turned off too much	3
Update to <code>xr</code>	3
Column data for <code>multicols*</code> sometimes vanished	3
Extension to <code>\docolaction</code> in <code>multicol</code>	3
Prevent color leak in <code>array</code>	3
Support fragile commands in <code>array</code> or <code>tabular</code> column templates	3
Changes to packages in the amsmath category	3
Website updates	3
Publications area reorganized and extended . .	3
Japanese translations of the user’s guide	3

Introduction

The December 2018 release of L^AT_EX is a maintenance release in which we have fixed a few bugs in the software: some are old, some newer, and they are mostly rather obscure.

Bug reports for core L^AT_EX 2_ε and packages maintained by the Project Team

In Spring 2018 we established a new issue tracking system (Github issues at <https://github.com/latex3/latex2e/issues>) for both the L^AT_EX core and the packages maintained by the L^AT_EX Project Team, with an updated procedure for how to report a bug or problem.

Initial experience with this system is good, with people who report problems following the guidelines and including helpful working examples to show the problem—thanks for doing this.

The detailed requirements and the workflow for reporting a bug in the core L^AT_EX software is documented at

<https://www.latex-project.org/bugs/>

with further details and discussion in [1].

Changes to the L^AT_EX kernel

UTF-8: updates to the default input encoding

In the April 2018 release of L^AT_EX we changed the default encoding from 7-bit ASCII to UTF-8 when using classic T_EX or pdfT_EX; see *L^AT_EX News 28* [2] for details.

Now, after half a year of experience with this new default, we have made a small number of adjustments to further improve the user experience. These include:

- Some improvements when displaying error messages about UTF-8 characters that have not been set up for use with L^AT_EX, or are invalid for some other reason; (*github issues 60, 62 and 63*)
- The addition of a number of previously missing declarations for characters that are in fact available with the default fonts, e.g., `\j` “j” (0237), `\SS` “SS” (1E9E), `\k{}` “_˘” (02DB) and `\.{}` “[˙]” (02D9);
- Correcting the names for `\guillemetleft` “«” and `\guillemetright` “»” in all encoding files. These correct names are in addition to the old (but wrong) Adobe names: Adobe mistakenly called them Guillemot, which is a sea bird. (*github issue 65*)
- Added `\Hwithstroke` (“H”) and `\hwithstroke` (“h”) necessary for typesetting Maltese. (<https://tex.stackexchange.com/q/460110>)

Fixed `\verb*` and friends in X_YL^AT_EX and LuaL^AT_EX

The original `\verb*` and `verbatim*` in L^AT_EX were coded under the assumption that the position of the space character (i.e., ASCII 32) in a typewriter font contains a visible space glyph “`␣`”. This is correct for pdfT_EX with the most used font encodings OT1 and T1. However, this unfortunately does not work for Unicode engines using the TU encoding since the space character slot (ASCII 32) then usually contains a real (normal) space, which has the effect that `\verb*` produces the same results as `\verb`.

The `\verb*` code now always uses the newly introduced command `\verbvisiblespace` to produce the visible space character and this command will get appropriate definitions for use with the different engines. With pdfT_EX it will simply use `\asciispace`, which is a posh name for “select character 32 in the current font”, but with Unicode engines the default definition is

```
\DeclareRobustCommand\verbvisiblespace
{\leavevmode
{\usefont{OT1}{cmtt}{m}{n}\asciispace}}
```

which uses the visible space from the font Computer Modern Typewriter, regardless of the currently chosen typewriter font. Internally the code ensures that the character used has exactly the same width as the other characters in the current (monospaced) font; thus, for example, code displays line up properly.

It is possible to redefine this command to select your own character, for example

```
\DeclareRobustCommand\verbvisiblespace
{\textvisiblespace}
```

will select the the “official” visible space character of the current font. This may look like the natural default, but it wasn’t chosen as our default because many fonts just don’t have that Unicode character, or they have one with a strange shape. (*github issues 69 and 70*)

Error message corrected

Trying to redefine an undefined command could in a few cases generate an error message with a missing space, e.g., `\renewcommand\1{...}` gave

```
LaTeX Error: \lundefined.
```

This is now fixed. (*github issue 41*)

Fixed fatal link error with hyperref

If an `\href` link text gets broken across pages, pdf \TeX and Lua \TeX will generate a fatal error unless both parts of the link are internally at the same boxing level. In two-column mode that was not the case if one of the pages had spanning top floats. This has now been changed so that the error is avoided. (*github issue 94*)

Avoid page breaks caused by invisible commands

Commands like `\label` or `\index` could generate a potential page break in places where a page break was otherwise prohibited, e.g., when used between two consecutive headings. This has now been corrected. If for some reason you really want a break and you relied on this faulty behavior, you can always add one using `\pagebreak`, with or without an optional argument.

(*github issue 81*)

Prevent spurious spaces when reading table of contents data

When table of contents data is read in from a `.toc` file, the new-line character at the end of each line is converted by \TeX to a space. In normal processing this is harmless (as \TeX is doing this input reading whilst in vertical mode and each line in the file represents a single line (paragraph) in the table of contents. If, however, this is done in horizontal mode, which is sometimes the case, then these spaces will appear in the output. If you then omit some of the input lines (e.g., because you do not display TOC data below a certain level), then these spaces accumulate in the typeset output and you get surprising, and unwanted, gaps inside the text.

The new code now adds a % sign at the end of problematic lines in the `.toc` file so that \TeX will not generate such spaces that may survive to spoil the printed result. As some third party packages have augmented or changed the core \LaTeX functionality in that area (for example, by adding additional arguments to the commands in TOC files) the code uses a conservative approach and the % signs are added only when certain conditions are met. Therefore some packages might require updates if they want to benefit from this correction, especially if they unconditionally overwrite \LaTeX ’s `\addcontentsline` definition. (*github issue 73*)

Prevent protrusion in table of contents lines

In \TeX ’s internal processing model, paragraph data is one of the major data structures. As a result, many things are internally modeled as paragraphs even if they are not conceptually “text paragraphs” in the traditional sense. In a few cases this has some surprising effects that are not always for the better. One example is standard TOC entries, where you have heading data followed by some dot leaders and a page number at the right, produced, for example, from this:

```
Error message corrected . . . . . 2
```

The space reserved for the page number is of a fixed width, so that the dots always end in the same place. Well, they did end in the same place until the advent of protrusion support in the \TeX engines. Now, with the `microtype` package loaded, it is possible that the page number will protrude slightly into the margin (even though it’s typeset inside a box) and as a result this page number box gets shifted. With enough bad luck this can get you another dot in the line, sticking out like the proverbial sore thumb, as exhibited in the question on StackExchange that triggered the correction.

\LaTeX now takes care that there will be no protrusion happening on such lines, even if it is generally enabled for the whole document.

(<https://tex.stackexchange.com/q/172785>)

Start L-R mode for \thinspace and friends

In \LaTeX , commands that are intended only for paragraph (L-R) mode are generally careful to start paragraph mode if necessary; thus they can be used at the start of a paragraph without surprising and unwanted consequences. This important requirement had been overlooked for a few horizontal spacing commands, such as `\thinspace` (a.k.a. “\,”), and for some other support commands such as `\smash` or `\phantom`. Thus they ended up adding vertical space when used at the beginning of a paragraph or, in the case of `\smash`, creating a paragraph of their own. This has now been corrected, and a corresponding update has been made to the `amsmath` package, in which these commands are also defined. (*github issues 49 and 50*)

Guarding \pfill in doc

For presenting index entries pointing to code fragments and the like, the `doc` package has a `\pfill` command

that generates within the index a line of dots leading from the command name to the page or code line numbers. If necessary it would automatically split the entry over two lines. That worked well enough for a quarter century, but we discovered recently that it is broken inside the `ltugboat` class, where it sometimes produces bad spacing within continuation lines.

The reason turned out to be a redefinition of the \LaTeX command `\nobreakspace` (\sim) inside the class `ltugboat`, which removed any preceding space (and thus unfortunately also removed the dots on the continuation line). While one can argue that this is a questionable redefinition (if only done by a single class and not generally), it has been in the class so long that changing it would certainly break older documents. So instead we now guard against that removal of space.

(*github issues 25 and 75*)

Changes to packages in the tools category

Sometimes the trace package turned off too much

The `trace` package is a useful little tool for tracing macro execution: it hides certain lengthy and typically uninteresting expansions resulting from font changes and similar activities. However, it had the problem that it also reset other tracing settings such as `\showoutput` in such situations, so that you couldn't use `\showoutput` in the preamble to get symbolic output of all the pages in the document. This has now been corrected.

Update to xr

The `xr` package has been updated so that the code that reads the `.aux` file has been made more robust. It now correctly ignores conditionals (added by `hyperref` and other packages) rather than generating low level parsing errors. (<https://tex.stackexchange.com/a/452321>)

Column data for multicols sometimes vanished*

In certain situations involving `multicols*`, when there are more explicit `\columnbreak` requests than there are columns on the current page, data could vanish due to the removal of an internal penalty marking the end of the environment. This has been corrected by explicitly reinserting that penalty if necessary. (*github issue 53*)

Extension to \docolaction in multicol

The `\docolaction` command can be used to carry out actions depending on the column you are currently in, i.e., first, any inner one (if more than two) or last. However, if the action generates text then there is the question: is this text part of the current column or the one after? That is, on the next run, do we test before or after it, to determine in which column we are?

This is now resolved as follows: if you use `\docolaction*` any generated text by the chosen action is considered to be after the test point. But if you use the command without the star then all the material it generates will be placed before the test point to determine the current column, i.e., the text will become part of the current column and may affect the test result on the next run.

Prevent color leak in array

In some cases the color used inside a `tabular` cell could “leak out” into the surrounding text. This has been corrected. (*github issue 72*)

Support fragile commands in array or tabular column templates

The preamble specifiers `p`, `m` and `b` each receives a user supplied argument: the width of the paragraph column. Normally that is something harmless, like a length or a simple length expression. But in more complicated settings involving the `calc` package it could break with a low-level error message. This has now been corrected.

(<https://tex.stackexchange.com/q/459285>)

Changes to packages in the amsmath category

The changes in the kernel made for `\thinspace`, `\smash`, etc. (see above) have been reflected in the `amsmath` package code, so that loading this package doesn't revert them. (*github issues 49 and 50*)

Website updates

Publications area reorganized and extended

To help readers to find relevant information in more convenient and easy ways, the area of the website covering publications by the \LaTeX Project Team was reorganized and extended (many more abstracts added). We now provide the articles, talks and supplementary data structured both by year and also by major topics [4]. Feel free to take a look.

Japanese translations of the user's guide

Yukitoshi Fujimura has kindly translated into Japanese two documents that are distributed with standard \LaTeX . These are:

- $\LaTeX 2_{\epsilon}$ for authors;
- User's Guide for the `amsmath` Package [5].

They can be found on the website documentation page [3]. You will now also find there a typeset version of the full $\LaTeX 2_{\epsilon}$ source code (with index etc.) and a number of other goodies.

References

- [1] Frank Mittelbach: *New rules for reporting bugs in the \LaTeX core software*. In: TUGboat, 39#1, 2018. <https://latex-project.org/publications/>
- [2] *\LaTeX News, Issue 28*. In: TUGboat, 39#1, 2018. <https://latex-project.org/news/latex2e-news/>
- [3] *\LaTeX documentation on the \LaTeX Project Website*. <https://latex-project.org/documentation/>
- [4] *\LaTeX Project publications on the \LaTeX Project Website*. <https://latex-project.org/publications/>
- [5] American Mathematical Society and The $\LaTeX 3$ Project: *User's Guide for the `amsmath` Package* (Version 2.1). April 2018. Available from <https://www.ctan.org> and distributed as part of every \LaTeX distribution.

Indexing, glossaries, and bib2gls

Nicola L. C. Talbot

Abstract

The `bib2gls` command line application [17] combined with the `glossaries-extra` package [18] provides an alternative indexing method to those provided by the base `glossaries` package [19]. In addition, since the terms are defined in `.bib` files, tools such as `JabRef` [3] can be used to manage large databases.

1 Introduction

The \LaTeX kernel provides two basic forms of indexing (that is, collating terms and their associated locations in the document). The first form creates a file called `\jobname.idx` using `\makeindex` and the information is written to the file using the command `\index{\langle info \rangle}`. This writes the line

```
\indexentry{\langle info \rangle}{\langle page \rangle}
```

to the `.idx` file, where `\langle page \rangle` is the page number.

The second form is very similar but uses a different file extension. A file called `\jobname.glo` is created with `\makeglossary` and the information is written using the command `\glossary{\langle info \rangle}` which writes the line

```
\glossaryentry{\langle info \rangle}{\langle page \rangle}
```

to the `.glo` file, where `\langle page \rangle` is the page number.

In both cases, the page number is obtained from `\thepage` and the write operation is delayed to ensure the value is correct in the event that the indexing occurs across a page break. These commands date back to the 1980s when processing power and resources were significantly smaller than today. Compilation of draft documents could be speeded up by omitting the indexing, which can be done by commenting out the `\make...` commands or by inserting `\nofiles` before them.

The next step is to collate the `\langle info \rangle``\langle page \rangle` information, removing duplicates, concatenating page ranges, hierarchically ordering the terms, and writing the \LaTeX code that will typeset the result as an index or glossary. \TeX isn't particularly suited for this kind of task. It's much more efficient to use a custom application; `makeindex` was created for this purpose, which creates a `.ind` file from the `.idx`.

2 Indexing

The \LaTeX kernel doesn't provide any specific commands for reading the file created by the indexing application, but instead defers this task to packages. The first of these was `makeidx` [5], which is quite trivial. It provides the command `\printindex` that inputs `\jobname.ind` if it exists and provides

commands for convenient 'see' and 'see also' cross-referencing.

2.1 Indexing example

A simple example that uses the `lipsum` package [2] for padding follows:

```
\documentclass{article}
\usepackage{lipsum}
\usepackage{makeidx}
\makeindex
\begin{document}
\index{duck}\lipsum*[1]\index{goose}
\par\lipsum[2-4]\par
\index{duck}\index{ant}\lipsum*[5-6]
\index{zebra}\par
\index{goose}\index{aardvark}\lipsum[7-10]\par
\lipsum*[11]\index{dog}\index{ant}\index{goose}
\printindex
\end{document}
```

If the file is called `myDoc.tex` then¹

```
latex myDoc
```

will create the file `myDoc.idx` that contains

```
\indexentry{duck}{1}
\indexentry{goose}{1}
\indexentry{duck}{1}
\indexentry{ant}{1}
\indexentry{zebra}{2}
\indexentry{goose}{2}
\indexentry{aardvark}{2}
\indexentry{dog}{3}
\indexentry{ant}{3}
\indexentry{goose}{3}
```

At this point the output doesn't contain an index, as the file `myDoc.ind` (which `\printindex` attempts to read) doesn't exist. This file can be created with

```
makeindex myDoc
```

The document then needs to be rerun to include the new `myDoc.ind` so the complete document build is

```
latex myDoc
makeindex myDoc
latex myDoc
```

The default behaviour of `makeindex` is to assume the extension `.idx` for the input file (if not specified) and use the extension `.ind` for the output file (which fits the expected extension used in `\printindex`). In this case, it creates a `myDoc.ind` containing (except the blank lines around each group are omitted, here and in the following):

```
\begin{theindex}
\item aardvark, 2
\item ant, 1, 3
```

¹ `latex` is used here to denote `pdflatex`, `xelatex`, etc., as appropriate.

```

\indexspace
\item dog, 3
\item duck, 1
\indexspace
\item goose, 1--3
\indexspace
\item zebra, 2
\end{theindex}

```

The `\indexspace` command usually produces a visual separation between letter groups. Neither that command nor the `\begin{theindex}` environment are provided by the \LaTeX kernel, so are defined by classes that provide index support.

If you try out this example, you'll find that `\index` doesn't produce any text where it's used in the document. This catches out some new users who expect the indexed term to also appear in the document. Typically, `\index` will be placed either before or after the appropriate word. For example:

```
Aardvarks\index{aardvark} are
\index{nocturnal animal}nocturnal animals.
```

The default output created by `makeindex` can be modified by a style file. For example, if I create a file called `myindexstyle.ist` that contains

```

headings_flag 1
heading_prefix "\\heading{"
heading_suffix "}"

```

and pass this file to `makeindex`:

```
makeindex -s myindexstyle.ist myDoc
```

then the resulting `myDoc.ind` will now contain

```

\begin{theindex}
\heading{A}
\item aardvark, 2
\item ant, 1, 3
\indexspace
\heading{D}
\item dog, 3
\item duck, 1
\indexspace
\heading{G}
\item goose, 1--3
\indexspace
\heading{Z}
\item zebra, 2
\end{theindex}

```

This custom command `\heading` will need to be defined somewhere in my document. A basic example:

```

\newcommand*{\heading}[1]{%
\item\textbf{#1}\par\nobreak\indexspace\nobreak}

```

Some newer, more sophisticated, classes (such as `memoir` [8]) and packages (such as `imakeidx` [1]) provide greater flexibility, making it easier to customize the index format.

The obsolete `glossary` package [12] provided an analogous version of `makeidx` designed for use with `\makeglossary` and `\glossary`. This was made more complicated by the need to provide some kind of separation between the term and its description to assist formatting, but it was essentially using the same kind of mechanism as the above indexing example. The `memoir` and `nomencl` [9] packages provide similar functions. As with `\index`, `\nomenclature` (as provided by `nomencl`) and `\glossary` do not produce any text where they're used in the document.

2.2 Indexing syntax

The `<info>` argument of both `\index` and `\glossary` needs to be given in the syntax of the indexing application used to process the data.² This catches out many new users who may still be learning \LaTeX syntax and don't realise that external tools may have different special characters.

For `makeindex`, the special characters are:

- The 'actual' character used to separate the sort value from the actual term when they're different (default: `@`).
- The 'level' character used to separate hierarchical levels (default: `!`).
- The 'encap' character used to indicate the page number encapsulating command (default: `|`).
- The 'quote' character used to indicate that the following character should be interpreted literally (default: `"`).

These characters can be changed in the `makeindex` style file, if required.

For example, using the defaults,

```
\index{deja vu@\emph{d\'ej\'a vu}}
```

This indicates that the term should be sorted as 'deja vu' but the term will be written to the output (`.ind`) file as

```
\emph{d\'ej\'a vu}
```

Up to three hierarchical levels are supported by `makeindex`, and also by the standard definition of `\begin{theindex}`:

```

\index{animal!nocturnal!owl}
\index{animal!nocturnal!aardvark}
\index{animal!crepuscular!ocelot}

```

This is converted by `makeindex` to

```

\item animal
\subitem crepuscular
\subsubitem ocelot, 1
\subitem nocturnal

```

² This refers to the kernel definitions of `\index` and `\glossary` which simply take one mandatory argument that's written to the relevant file.

```
\subsubitem aardvark, 1
\subsubitem owl, 1
```

(assuming the indexing occurred on page 1).

Each hierarchical level may have a separate sort and actual value. For example (except on one line),

```
\index{debutante@d'\ebutante!1945-1958@1945
\textendash1958}
```

Here, the top-level item has the sort value `debutante` (used by the indexing application to order the top-level entries) with the actual value `d'\ebutante` used for printing in the document's index.

The sub-item has the sort value `1945-1958` (used by the indexing application to order sub-entries relative to their parent entry) with the actual value `1945\textendash 1958` used within the document's index.

If a parent entry is also indexed within the document, it must exactly match its entry within the hierarchy. A common mistake is something like

```
\index{debutante@d'\ebutante}
\index{d'\ebutante!1945-1958@1945\textendash 1958}
```

In this case, `makeindex` treats `d'\ebutante` and `debutante@d'\ebutante` as two separate top-level entries, which results in an index where 'débutante' appears twice, and one entry doesn't have any sub-items while the other does.

The corresponding page number (location) can be encapsulated by a command using the `encap` special character. This should be followed by the command name without the leading backslash. For example, if on page 2 of my earlier example document I add an `encap` to the `aardvark` entry,

```
\index{aardvark|textbf}
```

then the resulting `.ind` file created by `makeindex` will now contain (`makeindex` inserts the `\`):

```
\item aardvark, \textbf{2}
```

Other commands could be included, for example,

```
\index{aardvark|bfseries\emph}
```

would end up in the `.ind` file as

```
\item aardvark, \bfseries\emph{2}
```

but obviously this would lead to the undesired effect of rendering the rest of the index in bold. A better solution is to define a semantic command that performs the required font change, such as

```
\newcommand*{\primary}[1]{\textbf{\emph{#1}}}
```

If the encapsulating command takes more than one argument, the final argument needs to be the page number and the initial arguments need to be added to the `encap`. For example,

```
\index{aardvark|textcolor{blue}}
```

The `makeidx` package provides two commands that can be used in this way:

```
\newcommand*\see[2]{\emph{\seename} #1}
\providecommand*\seealso[2]{\emph{\alsoname} #1}
```

(The `\seename` and `\alsoname` macros are language-sensitive commands that produce the text 'see' and 'see also'.) These commands both ignore the second argument, which means that the page number won't be displayed. This provides a convenient way of cross-referencing. For example, if on page 2 I have

```
\index{ant-eater|seealso{aardvark}}
```

then the `.ind` file would contain

```
\item ant-eater, \seealso{aardvark}{2}
```

From `makeindex`'s point of view, this is just another encapsulating command, so if I add

```
\index{ant-eater}
```

to page 1 and page 8, then this would lead to a rather odd effect in the index:

```
\item ant-eater, 1, \seealso{aardvark}{2}, 8
```

The page list now displays as '1, *see also* aardvark, 8'.

The simple solution is to place all the cross-referenced terms before `\printindex`. (`makeidx` closes the indexing file at the start of `\printindex`, which means that indexing can't take place after it.)

The `encap` value may start with (or) to indicate the start or end of an explicit range. If used, these must match. For example, on page 2:

```
\index{aardvark|(textbf}
```

and then on page 10:

```
\index{aardvark|)textbf}
```

results in

```
\item aardvark \textbf{2--10}
```

If no formatting is needed, (and) may be used alone:

```
\index{aardvark|}
```

```
...
```

```
\index{aardvark|)}
```

Although these parentheses characters have a special meaning at the start of the `encap`, they're not considered special characters.

If any of the special characters need to be interpreted literally, then they must be escaped with the quote character. For example,

```
\index{n"!@#n"!$}
```

```
\index{x@$"|\vec{x}"|$}
```

In the first case above, the special character that needs to be interpreted literally is the level character `!` which appears in both the sort value and the actual value. In the second case, the special character that needs to be interpreted literally is the `encap` character `|` which appears twice in the actual value. (Of course, replacing `|` with `\vert` avoids the problem.)

This is something that often trips up new users. With experience, we may realise that providing semantic commands can hide the special characters from the indexing application. For example,

```
\newcommand*\factorial}[1]{#1!}
```

This can take care of the actual value but not the sort value, which still includes a special character:

```
\index{n"!@$factorial{n}$}
```

The quote character itself also needs escaping if it's required in a literal context:

```
\index{naive@na""\i ve}
```

From `makeindex`'s point of view the backslash character `\` is a literal backslash so `\"` is a backslash followed by a literal double-quote (which has been escaped with `\"`).

2.3 UTF-8

So far, all examples that include accented characters have used accent commands, such as `\'`, since `makeindex` doesn't support UTF-8. This is essentially down to its age, as it was written in the mid-1980s, before the advent of Unicode. A previous *TUGboat* article [13] highlights the problem caused when trying to use `makeindex` on a UTF-8 file.

Around 2000, `xindy` [4], a new indexing application written in Perl and Lisp, was developed as a language-sensitive, Unicode-compatible alternative to `makeindex`. The native `xindy` format is quite different from the `makeindex` syntax described above and can't be obtained with `\index`. In this case, the special characters are the double-quote `"` used to delimit data and the backslash `\` used to indicate that the following character should be taken literally.

In the earlier factorial example, the `makeindex` syntax (used in the `.idx` file) is

```
\indexentry{n"!@$factorial{n}$}{1}
```

(assuming `\index{n"!@$factorial{n}$}` occurred on page 1). Whereas in the native `xindy` format this would be written as

```
(indexentry
 :tkey (("n!" "$\factorial{n}$"))
 :locref "1")
```

or

```
(indexentry
 :key ("n!")
 :print ("$\factorial{n}$")
 :locref "1")
```

The exclamation mark doesn't need escaping in this case but the backslash does. The `na""\i ve` example above needs both the backslash and double-quote treated in a literal context:

```
(indexentry
```

```
 :tkey (("naive" "na""\i ve"))
 :locref "1")
```

Of course, in this case UTF-8 is preferable:

```
(indexentry :key ("naïve") :locref "1")
```

This format requires a completely different command than `\index` for use in the document. However, `xindy` is capable of reading `makeindex` syntax. The simplest way of enabling this is by invoking `xindy` through the wrapper program `texindy`. Unfortunately, unlike `makeindex`, there's no way of changing the default special characters. The previous *TUGboat* article on testing indexing applications [13] compares `makeindex` and `xindy`.

A recent alternative that's also Unicode compatible is the Lua program `xindex` [20]. This reads `makeindex` syntax and command line switches are available to change the special characters or to specify the language.

2.4 Shortcuts

The `\index` command doesn't generate any text. This can lead to repetition in the code. For example,

```
An aardvark\index{aardvark} is a
nocturnal animal\index{nocturnal animal}.
```

It's therefore quite common to see users provide their own shortcut command to both display and index a term. For example,

```
\newcommand*\Index}[1]{#1\index{#1}}
%...
An \Index{aardvark} is a
\Index{nocturnal animal}.
```

Complications arise when variations are required. For example, if a page break occurs between 'nocturnal' and 'animal', so that 'nocturnal' is at the end of, say, page 1 and 'animal' is at the start of page 2, then placing `\index` after the term leads to the page reference 2 in the index whereas placing it before leads to the page reference 1. Also `\index` creates a whatsit that can cause interference. Although examples quite often place `\index` after the text, in many cases it's more appropriate to put `\index` first. This shortcut command doesn't provide the flexibility of the placement of `\index` relative to the text.

A problem also arises if the term includes special characters that need escaping in `\index` but not in the displayed text or if the display text needs to be a slight variation of the indexed term. For example, the above definition of `\Index` can't be used in the following:

```
The na""\i ve\index{naive@na""\i ve}
geese\index{goose} were frightened by the
flock of ph\oe nixes\index{phoenix@ph\oe nix}.
```

The definition of `\Index` could be modified to include an optional argument to provide a different displayed term. For example:

```
\newcommand*{\Index}[2][\thedisplaysymbol]{%
  \def\thedisplaysymbol{#2}%
  #1\index{#2}}
```

but this ‘shortcut’ ends up with slightly longer code:

```
The \Index[na\i ve]{naive@na\i ve}
\Index[geese]{goose} were frightened by the
flock of \Index[ph\oe nixes]{phoenix@ph\oe nix}.
```

An obvious solution to the first case (naïve) is to use UTF-8 instead of L^AT_EX accent commands combined with a Unicode-aware indexing application (`texindy` or `xindex`).

```
The \Index{naïve} geese\index{goose}
were frightened by the flock of
phœnixes\index{phœnix}.
```

This works fine with a Unicode engine (X_YL^AT_EX or LuaL^AT_EX) but not with `inputenc` [6], which uses the so-called active first octet trick to internally apply accent commands. This means that the `.idx` file ends up with

```
\indexentry{na\IeC {\i }ve}{1}
\indexentry{goose}{1}
\indexentry{ph\IeC {\oe }nix}{1}
```

The `\index` mechanism is designed to write its argument literally to the `.idx` file. This can be seen from the earlier `\factorial` example where

```
$$\factorial{n}$$\index{$$\factorial{n}$$}
```

is written to the `.idx` file as

```
\indexentry{$$\factorial{n}$$}{1}
```

Unfortunately, embedding `\index` in the argument of another command (such as the above custom `\Index`) interferes with this. For example,

```
\Index{$$\factorial{n}$$}
```

results in the expansion of `\factorial` as the indexing information is written to the `.idx` file:

```
\indexentry{!n!}{1}
```

The level special character (!) is no longer hidden from the indexing application and, since it hasn’t been escaped with the quote character, this leads to an unexpected result in the `.ind` file:

```
\item $n
  \subitem $, 1
```

2.5 Consistency

With `makeindex`, invisible or hard to see differences in the argument of `\index` can cause seemingly duplicate entries in the index. For example (line breaks here are part of the input),

```
\index{debutante@d'\ebutante
```

```
!1945-1958@1945\textendash 1958}
```

```
%...
```

```
\index
```

```
{debutante@d'\ebutante!1945-1958@1945\textendash
1958}
```

Here the two entries superficially appear the same but the line break inserted into the first instance results in two different entries in the index. The first entry has a space at the end of its actual value but the second doesn’t. This is enough to make them appear different entries from `makeindex`’s point of view. When viewing the `.ind` file, the difference is only perceptible if the text editor has the ability to show white space.

The simplest solution here is to run `makeindex` with the `-c` option, which compresses intermediate spaces and ignores leading and trailing blanks and tabs.

A long document with a large index of hierarchical terms and terms that require a non-identical sort value can be prone to such mistakes. Other inconsistencies can arise through misspellings (which hopefully the spell-checker will detect) or more subtle errors that are missed by spell-checkers.

For example, in English some words are hyphenated (‘first-rate’), some are merged into a single word (‘firstborn’) and some are space-separated (‘first aid’). Even native speakers can mix up the separator, and this can result in inconsistencies in a large document. For example,

```
\index{firstborn}
%...
\index{first-born}
%...
\index{first born}
```

From the spell-checker’s point of view, there are no spelling errors to flag in the above code. The inconsistencies can be picked up by proof-reading the index, but unfortunately some authors skip the back matter when checking their document.

When using `\glossary` (rather than `\index`), which may additionally include a long description, the problem with consistency becomes more pronounced. The example document below illustrates the use of the kernel version of `\glossary`, with one regular entry and one range entry. All the strings have to be exactly the same.

Also shown here is that since `makeindex` is a trusted application, it can be run through the shell escape in restricted mode. Finally, a `makeindex` style file is needed to indicate that data is now marked up with `\glossaryentry` instead of `\indexentry`:

```
\documentclass{report}
\begin{filecontents*}{\jobname.ist}
```

```

keyword "\\glossaryentry"
\end{filecontents*}

\IfFileExists{\jobname.glo}
{\immediate\write18{makeindex -s \jobname.ist
                    -o \jobname.gls \jobname.glo}}
{\typeout{Rerun required.}}

\makeglossary
\begin{document}
\chapter{Introduction}
Duck\glossary{duck: a waterbird
with webbed feet}\ldots

\chapter{Ducks}
\glossary{duck: a waterbird
with webbed feet|{}
\ldots
\glossary{duck: a waterbird
with webbed feet|)}

\renewcommand{\indexname}{Glossary}
\makeatletter
\@input@{\jobname.gls}
\makeatother
\end{document}

```

The old `glossary` package introduced a way of saving the glossary information and referencing it by label to perform the indexing, which helped consistency and reduced document code. The term could then be just indexed by referencing the label with `\useglossentry`, or could be both indexed and displayed with `\gls{<label>}`. Special characters still needed to be escaped explicitly, and this caused a problem for `\gls` as the quote character ended up in the document text. Abbreviation handling was performed using a different indexing command, and the package reached the point where it far exceeded its original simplistic design. It was time for a completely new approach, which we turn to now.

3 The glossaries package

The `glossaries` package [19] was introduced in 2007 as a replacement to the now obsolete `glossary` package. The main aims were to

- define *all* terms so that they can be referenced by label (no document use of `\glossary`);
- internally escape indexing special characters so that the user doesn't need to know about them;
- make abbreviations use the same indexing mechanism for consistency.

The advantage of first defining terms so that they can be referenced by label isn't only to help consistency but also improves efficiency. When a term is defined, partial indexing information is constructed and saved.

This is the point where any special characters are escaped, which means that this operation only needs to be done when the term is defined, not every time the term is indexed.

A hierarchical term is defined by referencing its parent by label; thus, the parent's indexing data can easily be obtained and prefixed with the level separator at the start of the child's data.

With the old `glossary` package, a term had only an associated name, description and sort value, but the new `glossaries` package provides extra fields, such as an associated symbol or plural form. Unfortunately, when developing the new package I was still thinking in terms of the old package that needed to include the name and description in the indexing information so that it could be displayed in the glossary. Early versions of the `glossaries` package continued this practice and the 'actual' part of the indexing information included the name, description and symbol, written to the indexing file in the form `\glossaryentryfield{<label>}{<name>}{<description>}{<symbol>}`

This caused a number of problems. First, the name, description and symbol values all had to be parsed for indexing special characters, which added to the document build processing time (especially for long descriptions). Second, long descriptions could cause the indexing information to exceed `makeindex`'s buffer.

The package settings can allow for expansion to occur when terms are defined (for example, if terms are defined in a programmatic context that uses scratch variables that need expanding). This can lead to robust internal commands appearing in the indexing information. To simplify the problem of trying to escape all the `@` characters, the `glossaries` package uses the question mark character (?) as the actual character instead.

While it was necessary with the `glossary` package to write all this information to the indexing file, it is no longer necessary with the `glossaries` package as the name, description and symbol can all now be accessed by referencing the corresponding field associated with the term's identifying label. Therefore, newer versions now simply use

```
\glossentry{<label>}
```

for the actual text. Hierarchical entries use

```
\subglossentry{<level>}{<label>}
```

for the actual text, where `<level>` is the hierarchical level that's calculated when the term is defined. (This information may be of use to glossary styles that support hierarchical entries.) It's now quicker to construct the indexing information and only the sort value and label need checking for special characters.

For example,

```
\documentclass{report}
\usepackage[colorlinks]{hyperref}
\usepackage[symbols,style=treetgroup]{glossaries}
\makeglossaries

\newglossaryentry{waterbird}% label
{name={waterbird},
description={bird that lives in or near water}}

\newglossaryentry{duck}% label
{name={duck},
parent={waterbird},
description={a waterbird with webbed feet}}

\newglossaryentry{goose}% label
{name={goose},
plural={geese},
parent={waterbird},
description={a waterbird with a long neck}}

\newglossaryentry{fact}% label
{name={\ensuremath{n!}},
description={\$n\$ factorial},
sort={n!},
type=symbols
}
\begin{document}
\chapter{Singular}
\Gls{duck} and \gls{goose}.
\chapter{Plural}
\Glspl{duck} and \glspl{goose}.
\chapter{Other}
\begin{equation}
\gls[counter=equation]{fact} = n \times (n-1)!
\end{equation}
\printglossaries
\end{document}
```

This uses `\makeglossaries` (provided by `glossaries`), rather than `\makeglossary`, as it's not simply opening one associated file. The `glossaries` package supports multiple glossaries and all associated files are opened by this command as well as the custom style file for use by the indexing application. In this case, the document has two glossaries: the default main glossary and the symbols list (created with the `symbols` package option).

The glossaries are output using the command `\printglossaries`. This is a shortcut to iterate over all defined glossaries, calling for each

```
\printglossary[type=<label>]
```

where `<label>` identifies the required glossary. It's this command that inputs the file generated by the indexing application. A style is needed that supports hierarchical entries. In this example, I've chosen the `tree` style in the package options but the

style can also be set within the optional argument of `\printglossary`. (A list of all styles with example output can be viewed at the [glossaries gallery](#) [15].)

As shown here: the `hyperref` package must be loaded before `glossaries`. This is an exception to the general rule that `hyperref` should be loaded last.

The commands `\gls`, `\Gls`, `\glspl` and `\Glspl` all reference a term by its label and simultaneously display and index the term. The variations provide a way of displaying the plural form (`\glspl` and `\Glspl`) and to convert the first letter to upper case (`\Gls` and `\Glspl`). In this case, the 'waterbird' entry isn't explicitly indexed in the document but it's included in the indexing information for its child entries 'duck' and 'goose', which are indexed.

The above example creates two indexing files with extensions `.glo` (for the default glossary) and `.slo` (for the symbols list), that both use `makeindex` syntax. Each file contains the indexing information for a particular glossary. Both files require the `.ist` style file that's also created during the document build.

The lines are quite long but are all in the form (line breaks for clarity)

```
\glossaryentry
{<data>|<encap>}
{<location>}
```

The `<encap>` and `<location>` information can vary with each indexing instance but the `<data>` part is constant for each term, and it's this part that's created when the term is defined.

When the 'waterbird' term is defined, the `<data>` part is determined to be

```
waterbird?\glossentry{waterbird}
```

The sort part here is `waterbird` and the actual part is `\glossentry{waterbird}`. This is stored internally and accessed when the child entries are defined. For example, when the 'duck' entry is defined, its `<data>` information is set to (line break for clarity and not included in `<data>`)

```
waterbird?\glossentry{waterbird}!
duck?\subglossentry{1}{duck}
```

The hierarchical level numbering starts with 0 for top-level entries, so the duck entry has the level set to 1 since it has a parent but no grandparent.

The factorial example has this `<data>` part set:

```
n"!?\glossentry{fact}
```

Note that the special character occurring in the sort value has been escaped. This has to be done only once, when the entry is defined.

The location number defaults to the page number but may be changed, as in the reference to the `fact` entry, which switches to the `equation` counter:

```
\begin{equation}
\gls[counter=equation]{fact} = n \times (n-1)!
\end{equation}
```

Since the `report` class is in use, this is in the form `\langle chapter \rangle . \langle equation \rangle` (3.1 in this case).

Location formats only have limited support with `makeindex`, which requires a bare number (0, 1, 2, ...), Roman numeral (i, ii, iii, ... or I, II, III, ...), basic Latin letter (a, ..., z or A, ..., Z) or a simple composite that combines these forms with a given separator (such as A-4 or 3.1). The separator must be consistent, so you can't have a mixture of, say, A:i and B-2 and 3.4.

In this case, the separator is a period or full stop character, which is the default setting in the custom style file created by `\makeglossaries`, so `makeindex` will accept '3.1' as a valid location.

Unfortunately, the glossary style may need to know which counter generated each location. This is especially true if the `hyperref` package is in use and the location numbers needs to link back to the corresponding place in the document. The hyperlink information can't be included in the indexed location as it will be rejected as invalid by `makeindex`. The only other part of the indexing information that can vary without `makeindex` treating the same term as two separate entries is within the `encap`, so the `glossaries` package actually writes the `encap` as

```
setentrycounter[\langle h-prefix \rangle]{\langle counter \rangle}\langle csname \rangle
```

where `\langle counter \rangle` is the counter name and `\langle csname \rangle` is the name of the actual encapsulating command. This defaults to `glsnumberformat` but may be changed in the optional argument of commands like `\gls`.

The first example document at the start of this article demonstrated `makeindex`'s implicit range formation, where the location list for the 'goose' entry (which was indexed on pages 1, 2 and 3) was compressed into 1--3. This compression can only occur if the `encap` is identical for each of the indexing instances within the range.

The `hypertarget` will necessarily change for each non-identical indexed location. This means that if the actual target is included in the `encap` it will interfere with the range formation. Instead, only a prefix is stored (`\langle h-prefix \rangle`) which can be used to reconstruct the `hypertarget`. This assumes that `\theH\langle counter \rangle` is defined in the form `\langle h-prefix \rangle \the\langle counter \rangle`. Now the `encap` will be identical for identical values of `\langle h-prefix \rangle`. If the `hypertarget` can't be reconstructed from the location by simply inserting a prefix then it's not possible to have hyperlinked locations with this indexing method.

In the above example, the `report` class has been loaded along with `hyperref` so `\theHequation` is defined as

```
\theHsection.\arabic{equation}
```

This means that the indexing of the term in equation 3.1 occurs when `\theHequation` expands to 3.0.1 (the section counter is 0) so `\langle h-prefix \rangle` can't be obtained since there's no prefix that will make `\langle prefix \rangle 3.1` equal to 3.0.1. This results in a warning from the `glossaries` package:

```
Hyper target `3.0.1' can't be formed by
prefixing location `3.1'. You need to modify
the definition of \theHequation otherwise
you will get the warning: "`name{equation.3.1}'
has been referenced but does not exist"
```

(and `hyperref` does indeed generate that warning once the glossary files have been created). The only solution here is to either remove the location hyperlink or redefine `\theHequation` so that a prefix can be formed.

3.1 xindy

Although the `glossaries` package was originally designed for use with just `makeindex`, version 1.17 added support for `xindy`. It made sense to use `xindy`'s native format as it's more flexible; also, `texindy` only accepts the default `makeindex` special characters so it won't accept ? as the actual character.

The default setting assumes the `makeindex` application will be used for backward compatibility. The `xindy` package option will switch to `xindy` syntax. Again the partial indexing data is constructed when each entry is defined, but now the special characters that need escaping are " and \.

The previous example can be converted to use `xindy` by modifying the package options:

```
\usepackage[symbols,style=treegroup,xindy]
{glossaries}
```

The package also needs to know which counters (aside from the default `page` counter) will be used for locations. In our example, since one of the terms is indexed with the `equation` counter, this needs to be indicated:

```
\GlsAddXdyCounters{equation}
```

(The argument should be a comma-separated list if you are indexing other counters as well.)

As with `makeindex`, it's not straightforward to add the information needed to convert the location into a hyperlink. Now the prefix and location are provided using

```
:locref "\langle h-prefix \rangle{\langle location \rangle}"
```


and the counter and encapsulating format are merged into the attribute value:

```
:attr "<counter>(format)"
```

This is why with the `xindy` package option set it's necessary to specify which non-default counters and formats you want to use—so that corresponding commands can be provided.

Unlike `makeindex`, which will only accept very specific types of numbering, with `xindy` you can have your own custom numbering scheme, provided that you define a location class that specifies the syntax. This is obviously a far more flexible approach but the downside is a far greater chance that the location might include `xindy`'s special characters that will need escaping, and now the escaping must be done every time an entry is indexed, not just when the entry is defined.

Suppose for example, I have a little package (that loads `etoolbox` [7] and `tikzducks` [11]) that provides the robust command `\ducknumber{<n>}` to display `<n>` little ducks in a row,

```
\newcount\duckctr
\newrobustcmd{\ducknumber}[1]{%
  \ifnum#1>0\relax
    \duckctr=0\relax
    \loop
      \advance\duckctr by 1\relax
      \tikz[scale=0.3]{\duck;}%
    \ifnum\duckctr<#1
      \repeat
  \fi
}
```

It also provides `\duckvalue{<counter>}` if the value needs to be obtained from a counter:

```
\newcommand*\duckvalue[1]{%
  \ducknumber{\value{#1}}
```

Now let's suppose I want the page numbering in my document to be represented by ducks:

```
\renewcommand{\thepage}{\duckvalue{page}}
```

so, for example, on page 5, five little ducks are displayed in the footer. Now let's suppose that I index a term on this page. The location will expand to

```
\ducknumber{5}
```

This would be rejected as invalid by `makeindex`, but what about `xindy`? With an appropriate location class `xindy` would accept this, but it would interpret `\d` as the literal character 'd'. The resulting code it would write to the designated output file would be `ducknumber{5}`

so you'd end up with 'ducknumber5' typeset in your document.

The backslash must be escaped but there's a conflict between expansion and \TeX 's asynchronous

output routine. With the `glossaries` package, the location is obtained by expanding the command `\theglentrycounter`, and the corresponding hypertarget value (if supported) is obtained by expanding `\theHglentrycounter`. These two commands can be fully expanded when trying to determine the prefix. If the value of the `page` counter is currently wrong, then it's equally wrong for both values and it should still be possible to obtain the prefix.

When it comes to the actual task of preparing the location so that it's in a suitable format for `xindy`, there's no sense in converting `\theglentrycounter` into `\\theglentrycounter` as clearly there's no way for `xindy` to extract the page number from this. On the other hand, if `\theglentrycounter` is fully expanded (and then detokenized and escaped), the page number could end up incorrect if it occurs across a page break.

The normal way around this problem (used by `\protected@write`) is to locally let `\thepage` to `\relax` so that it isn't expanded until the actual write operation is performed, but if this method is used the location will end up as `\\thepage` which will prevent `xindy` from obtaining the correct value.

It's necessary for `\thepage` to be expanded before the write operation in order to escape the special characters but at the same time, the actual value of `\c@page` shouldn't be expanded until the write operation is actually performed.

Essentially, for the duck numbering example, on page 5 `\thepage` needs to be converted into

```
\\ducknumber{\the\c@page}
```

where `\\` are two literal (catcode 12) characters and `\the\c@page` is left to expand when the write operation is performed.

The `glossaries` package gets around this problem with a nasty hack that locally redefines some commands. For example, `\@alph\c@page` expands to `\gls@alphpage`. This command is skipped when the special characters are escaped but expands to the original definition of `\@alph\c@page` when the write operation is actually performed.

This action is only performed when the `page` counter is being used for the location. Other counters will need to be expanded immediately to ensure that they are the correct value.

As this hack can cause problems in some contexts, if you know that your locations will never expand to any content that contains `xindy` special characters, then it's best to switch off this behaviour with the package option `esclocations=false`.

This is an inherent problem when converting from one syntax (\LaTeX in this case) to another

(xindy or makeindex). Each syntax has its own set of special characters (required to mark up or delimit data) that may need to be interpreted literally.

3.2 Using T_EX to sort and collate

Some users who aren't familiar with command line tools have difficulty integrating them into the document build and prefer a T_EX-only solution that doesn't require them. In general, it's best to use tools for the specific task they were designed for. Indexing applications are designed for sorting and collating data. T_EX is designed for typesetting. Each tool is optimized for its own particular intended purpose. It is possible to sort and collate in T_EX but it's much less efficient than using a custom indexing application. However, for small documents it may suit some users to have everything done within T_EX, so version 4.04 of the `glossaries` package introduced a T_EX-only method.

The example document given on page 53 can be converted to use this method simply by replacing `\makeglossaries` with `\makenoidxglossaries` and `\printglossaries` with `\printnoidxglossaries`. As with `\printglossaries`, this is a shortcut command that iterates over all defined glossaries, doing `\printnoidxglossary[type=(label)]`

In this case, the command doesn't input a file but sorts the list of entry labels and iterates over them to display the information using the required glossary style. The label list only includes those entries that have been indexed in the previous L^AT_EX run. This information is obtained from the `.aux` file. Each time an entry is indexed using commands like `\gls`, a line is written to the `.aux` file in the form

```
\gls@reference{<type>}{<label>}{<location>}
```

where `<type>` identifies the glossary, `<label>` identifies the entry and `<location>` is in the form

```
\glsnoidxdisplayloc{<h-prefix>}{<counter>}{<encap>}{<number>}
```

This has the advantage that there is no conversion from one syntax to another and there's no restriction on `<number>` (as long as it's valid L^AT_EX code). The disadvantages are that there's no range support and sorting is slow and uses character code comparisons. (See my earlier *TUGboat* article comparing indexing methods [13].)

With this method, each entry has an associated internal field labelled `loclist`. When the `.aux` file is parsed, each location is added to this field using one of `etoolbox`'s internal list commands. This list is iterated over in order to display the locations.

4 The `glossaries-extra` package

The `glossaries-extra` package [18] was created in 2015 as a compromise between the conflicting requirements of users who wanted new features and users who complained that the `glossaries` package took a long time to load (because it had so many features). New features, especially those that require additional packages, necessarily add to the package load time.

The `glossaries-extra` package automatically loads the base `glossaries` package, but there are some differences in the default settings, the most noticeable being the abbreviation handling. The base package only allows one abbreviation style to be used throughout the document. The extension package defines a completely different mechanism for handling abbreviations that allows multiple styles within the same document.

As with the base package, the default indexing application is still assumed to be `makeindex` but the extension package provides two extra methods (although from L^AT_EX's point of view they both use the same essential code).

The new command `\printunsrtglossary[<options>]`

works fairly similarly to `\printnoidxglossary`, in that it iterates over a list of labels, but the list contains all the labels defined in the given glossary (rather than just those that have been indexed) and no sorting is performed by T_EX.

As with the other methods, there's a shortcut command that iterates over all glossaries:

```
\printunsrtglossaries
```

For example,

```
\documentclass{report}
\usepackage[colorlinks]{hyperref}
\usepackage[symbols,style=treegroup]
{glossaries-extra}
```

```
\newglossaryentry{waterbird}{name={waterbird},
description={bird that lives in or near water}}
```

```
\newglossaryentry{duck}{name={duck},
parent={waterbird},
description={a waterbird with webbed feet}}
```

```
\newglossaryentry{goose}{name={goose},
plural={geese},
parent={waterbird},
description={a waterbird with a long neck}}
```

```
\newglossaryentry{fact}{name={\ensuremath{n!}},
description={\$n\$ factorial},
sort={n!},
type=symbols
}
```

```
\begin{document}
\printunsrtglossaries
\end{document}
```

No indexing is performed in this document. With the other methods provided by the base package this would result in empty glossaries, but with this method all defined entries are shown (and only one L^AT_EX call is required to display the list). The ‘goose’ entry appears after ‘duck’ but only because ‘goose’ was defined after ‘duck’.

The glossary style I’ve chosen here (`treegroup`) shows the letter group headings. This is something that’s usually determined by the indexing applications according to the first character of the sort value. The heading information is then written to indexing output file (read by `\printglossary`) at the start of a new letter block.

The ‘noidx’ method checks the first letter of the sort value at the start of each iteration, and if it’s different from the previous iteration a new heading is inserted. The ‘unsrt’ method also does this unless the `group` key has been defined, in which case the letter group label is obtained from the corresponding field (if it’s set).

This letter group formation can lead to strange results if the entries aren’t defined in alphabetical order [16]. For example,

```
\documentclass{article}
\usepackage[style=treegroup]{glossaries-extra}

\newglossaryentry{ant}{name={ant},
description={small insect}}

\newglossaryentry{aardvark}{name={aardvark},
description={animal that eats ants}}

\newglossaryentry{duck}{name={duck},
description={waterbird with webbed feet}}

\newglossaryentry{antelope}{name={antelope},
description={deer-like animal}}

\begin{document}
\printunsrtglossaries
\end{document}
```

This produces the document shown in figure 1. (The vertical spacing below the letter headings is too large, but that is the default result; the point here is the undesired second ‘A’ group.)

If the `group` key has been defined but not explicitly set then an empty headerless group is assumed. If the above example is modified so that it defines the `group` key:

```
\glsaddstoragekey{group}{\grouplabel}
```

Glossary

A

ant small insect
aardvark animal that eats ants

D

duck waterbird with webbed feet

A

antelope deer-like animal

Figure 1: Example glossary with letter groups

Glossary

ant small insect
aardvark animal that eats ants
duck waterbird with webbed feet
antelope deer-like animal

Figure 2: Example glossary with empty group

but without modifying the entry definitions to set this key then no letter groups are formed (see figure 2).

The `record` package option automatically defines the `group` key. Each group value should be a label. The corresponding title can be set with

```
\glsxtrsetgrouptitle{<label>}{<title>}
```

For example,

```
\documentclass{article}
\usepackage[style=treegroup,record]
{glossaries-extra}
\glsxtrsetgrouptitle{antrelated}{Ants and
Ant-Eaters}
\glsxtrsetgrouptitle{waterbirds}{Waterbirds}
\glsxtrsetgrouptitle{deerlike}{Deer-Like}
\newglossaryentry{ant}{name={ant},
group={antrelated},
description={small insect}}
\newglossaryentry{aardvark}{name={aardvark},
group={antrelated},
description={animal that eats ants}}
\newglossaryentry{duck}{name={duck},
group={waterbirds},
description={waterbird with webbed feet}}
\newglossaryentry{antelope}{name={antelope},
group={deerlike},
description={deer-like animal}}
\begin{document}
\printunsrtglossaries
\end{document}
```

Glossary

Ants and Ant-Eaters

ant small insect
aardvark animal that eats ants

Waterbirds

duck waterbird with webbed feet

Deer-Like

antelope deer-like animal

Figure 3: Example glossary with custom groups

This now produces the glossary shown in figure 3. (Alternatively, use the `parent` key for a hierarchical structure or the `type` key to separate the logical blocks into different glossaries [16].)

`\printunsrtglossary` uses an iteration handler that supports the `loclist` internal field used with the ‘`noidx`’ method. If this field is set, the locations will be displayed but, as with the ‘`noidx`’ method, no ranges are formed and the elements of the `loclist` field must conform to a specific syntax. However, the handler will first check if the `location` field is set. If it is, that will be used instead.

The `location` key isn’t provided by default but is defined by the `record` option, so locations can also be provided when a term is defined. For example,

```
\newglossaryentry{ant}{name={ant},
  group={antrelated},
  location={1, 4--5, 8},
  description={small insect}}
```

This may seem cumbersome to do manually but it’s the underlying method used by `bib2gls` [17].

5 Glossaries and `.bib`: `bib2gls`

Some years ago I was asked if it was possible to provide a GUI (graphical user interface) application to manage files containing many entry definitions. This article has only mentioned defining entries with `\newglossaryentry` but there are other ways of defining terms with the `glossaries` package (and some additional commands provided with `glossaries-extra`). I already have several GUI applications that are quite time-consuming to develop and maintain, and the proposed task seemed far too complex, so I declined.

More recently, a question was posted on StackExchange [10] asking if it was possible to store terms in a `.bib` file, which could be managed in an application such as JabRef [3], and then converted into a `.tex` file containing commands such as `\newglossaryentry`.

This was a much better proposition as the graphical task could be dealt with by JabRef and the conversion tool could be a command line application.

I added the `record` option and the commands like `\printunsrtglossary` to `glossaries-extra` to assist this tool. The `record` option not only creates new keys (`group` and `location`) but also makes references to undefined entries trigger warnings rather than errors. This is necessary since the entries won’t be defined on the first \LaTeX call. The option also changes the indexing behaviour. As with the ‘`noidx`’ method, the indexing information is written to the `.aux` file so that the new tool could find out which entries are required and their locations in the document. In this case, the `.aux` entry is in the form

```
\glxtr@record{<label>}{<h-prefix>}{<counter>}
{<encap>}{<location>}
```

As with the ‘`noidx`’ method there is no conversion from one syntax to another when the indexing takes place, so there is no need to worry about escaping special indexing characters.

It later occurred to me that, without the constraints of the `makeindex` or `xindy` formats, it’s possible to save the `hypertarget` so that it doesn’t have to be reconstructed from `<h-prefix>` and `<location>`. In `glossaries-extra` version 1.37 I added the package option `record=nameref`, which writes more detailed indexing information to the `.aux` file (and support for this new form was added to `bib2gls` v1.8). This means that the earlier `makeindex` example on page 53 can be rewritten in such a way that the equation location now has a valid hyperlink.

\TeX syntax can be quite hard to parse programmatically. Regular expressions don’t always work. I have a number of applications that are related to \TeX in some way and need to parse either complete documents or code fragments. The most complicated of these was a Java GUI application used to assist production editors. The document code submitted by authors often contained problematic code that needed fixing, which was both tedious and time-consuming, so I tried to develop a system that parsed the original source provided by the authors and created new files with the appropriate patches and comments alerting the production editors of a potential problem, where necessary. The files were also flattened (that is, `\input` was replaced by the contents of the referenced file) to reduce clutter.

I realised that the \TeX parsing code used in this application would also be useful in some of my other Java applications so, rather than producing unnecessary duplication, I split the code off into a separate library, `texparserlib.jar` [14]. Rather than testing the code in big GUI applications that take a long

time to set up and run, I added a small application called `texparserapp.jar` to the `texparser` repository together with a selection of sample files to test the library.

The production editor GUI application not only needed to parse the `.tex` and `.bib` files supplied by the authors but also needed to gather information from `.aux` files. Some of this information is displayed in the graphical interface and it looks better if \LaTeX commands like `\'` or `\c` are converted to Unicode when showing author names. It used to also be a requirement for production editors to produce HTML files containing the abstract. I originally used `TeX4ht` for this but an update caused a conflict, and since only the abstract needed converting and `MathJax` could be used for any mathematical content, I decided that the \TeX parser code should not only provide \LaTeX to \LaTeX methods but also \LaTeX to HTML — with the caveat that the conversion to HTML was not intended for complete documents but for code fragments supplemented by information obtained from the `.aux` file.

The GUI application was used not only to prepare workshop proceedings but also to prepare a related series of books that contained reprints. Since writing the \TeX parser library the requirements for the proceedings have changed, which make the production editing task easier, and the publisher for the related series has also changed and the new publisher provides their own templates. The application has now largely become redundant although it can still be used to prepare volumes for the proceedings.

Since I already had this library that was designed to obtain information from `.aux` and `.bib` files, it made sense to use it for my new tool. This meant that the new tool also had to be in Java. The library methods that can convert \LaTeX code fragments to HTML provide a useful way of obtaining an appropriate sort value from the `name` field as accent commands can be converted to Unicode characters. Command definitions provided in `@preamble` can also be interpreted (provided they aren't too complex). Any HTML markup is stripped and leading and trailing white space is trimmed. This means that there should rarely be any need to set the `sort` field when defining an entry.

Sorting can be performed according to a valid language tag, such as `en` (English) or `en-GB` (British English) or `de-CH-1996` (Swiss German new orthography). Java 8 has support for the Unicode Common Locale Data Repository (CLDR) which provides collation rules, so `bib2gls` can support more languages than `xindy` (although, unlike `xindy`, it doesn't support Klingon).

There are other sort methods available as well, including sorting according to Unicode value (case-sensitive or case-insensitive) or sorting numerically (assuming the sort values are numbers) or sorting according to use in the document (determined by the ordering of the indexing information contained within the `.aux` file).

For example, suppose the file `entries.bib` contains the following:

```
% Encoding: UTF-8
@entry{waterbird,
  name={waterbird},
  description={bird that lives in or near water}}
@entry{goose,
  name={goose},
  parent={waterbird},
  description={waterbird with a long neck}}
@entry{duck,
  name={duck},
  parent={waterbird},
  description={waterbird with webbed feet}}
```

and suppose the file `symbols.bib` contains

```
% Encoding: UTF-8
@preamble{"\providecommand{\factorial}[1]{#1!}
\providecommand{\veclength}[1]{|#1|}"}
@symbol{nfact,
  name={\ensuremath{\factorial{n}}},
  description={n$ factorial}}
@symbol{lenx,
  name={\ensuremath{\veclength{\vec{x}}}},
  description={length of $\vec{x}$}}
```

The document code

```
\documentclass{report}
\usepackage[colorlinks]{hyperref}
\usepackage[symbols,style=treegroup,
  record=nameref]
  {glossaries-extra}
\GlsXtrLoadResources[
  src=entries,% entries.bib
  sort=en-GB]
\GlsXtrLoadResources[
  src=symbols,% symbols.bib
  type=symbols,% glossary
  sort=letter-nocase]

\begin{document}
\chapter{Singular}
\Gls{duck} and \gls{goose}.
\chapter{Plural}
\Glspl{duck} and \glspl{goose}.
\chapter{Other}
\begin{equation}
\gls[counter=equation]{nfact} = n \times (n-1)!
\end{equation}
The length of  $\vec{x}$  is  $\gls{lenx}$ .\par
\printunsrtglossaries
\end{document}
```

Unlike the `makeindex` and `xindy` methods, which require one call per glossary, with this approach only one `bib2gls` call is required, regardless of the number of glossaries. For example, if the document code is in `myDoc.tex`, then the build process is

```
pdflatex myDoc
bib2gls myDoc
pdflatex myDoc
```

Letter groups are not formed by default. To get them, specify the `-g` switch:

```
bib2gls -g myDoc
```

`bib2gls` creates one `.glsstex` output file per instance of `\GlsXtrLoadResources`, but you don't necessarily need one `\GlsXtrLoadResources` per glossary. You may be able to process multiple glossaries within one instance of this command, or a single glossary may require multiple instances.

The `.glsstex` file contains the glossary definitions (using provided wrapper commands for greater flexibility) in the order obtained from the provided sort method. In the above example, the entries in the first `.glsstex` file are defined in the order obtained by sorting the values according to the `en-GB` rule. The entries in the second `.glsstex` file are defined in the order obtained by sorting the values according to the `letter-nocase` rule (that is, case-insensitive Unicode order).

If the `sort` key isn't provided (which it generally isn't), its value is taken from the designated fallback field. In the case of `@entry` this is the `name` field and in the case of `@symbol` this is the entry's `label`. So in the above example, the symbols are sorted as first '`lenx`' and second '`nfact`'.

The fallback field used for `@symbol` entries can be changed. For example, to switch to the `name` field:

```
\GlsXtrLoadResources [
  src=symbols,
  type=symbols,
  symbol-sort-fallback=name,
  sort=letter-nocase
]
```

Since the `name` field contains commands, the `TEX` parser library is used to interpret them. The transcript file (`.glg`) shows the results of the conversion. The `nfact` entry ends up with just two characters, '`n!`' but the `lenx` entry ends up with four characters: vertical bar (Unicode 0x7C), lower case 'x' (Unicode 0x78), combining right arrow above (Unicode 0x20D7) and vertical bar (Unicode 0x7C). The order is now: `n!` (`nfact`), `|x̄|` (`lenx`).

References

- [1] Claudio Beccari and Enrico Gregorio. The `imakeidx` package, 2018. ctan.org/pkg/imakeidx.

- [2] P. Happel. The `lipsum` package, 2019. ctan.org/pkg/lipsum.
- [3] JabRef: Graphical frontend to manage `BIBTEX` databases, 2018. jabref.org.
- [4] R. Kehr and J. Schrod. `xindy`: A general-purpose index processor, 2018. ctan.org/pkg/xindy.
- [5] The `LATEX` Team. The `makeidx` package, 2014. ctan.org/pkg/makeidx.
- [6] The `LATEX` Team, F. Mittelbach, and A. Jeffrey. The `inputenc` package, 2018. ctan.org/pkg/inputenc.
- [7] P. Lehman and J. Wright. The `etoolbox` package, 2018. ctan.org/pkg/etoolbox.
- [8] L. Madsen and P. R. Wilson. The `memoir` class, 2018. ctan.org/pkg/memoir.
- [9] L. Netherton, C. V. Radhakrishnan, et al. The `nomencl` package, 2019. ctan.org/pkg/nomencl.
- [10] The Pompitous of Love. Is there a program for managing glossary tags?, 2016. tex.stackexchange.com/questions/342544.
- [11] samcarter. The `tikzducks` package, 2018. ctan.org/pkg/tikzducks.
- [12] N. Talbot. The `glossary` package, 2006. ctan.org/pkg/glossary.
- [13] N. Talbot. Testing indexes: `testidx.sty`. *TUGboat* 38(3):377–399, 2017. tug.org/TUGboat/tb38-3/tb120talbot.pdf.
- [14] N. Talbot. `texparserlib.jar`: A Java library for parsing (`LA`)`TEX` files, 2018. github.com/nlct/texparser.
- [15] N. Talbot. Gallery of all styles provided by the `glossaries` package, 2019. dickimaw-books.com/gallery/glossaries-styles.
- [16] N. Talbot. Logical glossary divisions (type vs group vs parent), 2019. dickimaw-books.com/gallery/logicaldivisions.shtml.
- [17] N. Talbot. `bib2gls`: Command line application to convert `.bib` files to `glossaries-extra.sty` resource files, 2019. ctan.org/pkg/bib2gls.
- [18] N. Talbot. The `glossaries-extra` package, 2019. ctan.org/pkg/glossaries-extra.
- [19] N. Talbot. The `glossaries` package, 2019. ctan.org/pkg/glossaries.
- [20] H. Voß. `xindex`: Unicode compatible index generation, 2019. ctan.org/pkg/xindex.

◇ Nicola L. C. Talbot
 School of Computing Sciences
 University of East Anglia
 Norwich Research Park
 Norwich NR4 7TJ
 United Kingdom
 N.Talbot (at) uea dot ac dot uk
<http://www.dickimaw-books.com>

TeX.StackExchange cherry picking, part 2: Templating

Enrico Gregorio

Abstract

We present some examples of macros built with `expl3` in answer to users' problems presented on `tex.stackexchange.com`, to give a flavor of the language and explore its possibilities.

1 Introduction

This is the second installment¹ of my cherry picking from questions on TeX.SX that I answered using `expl3`. As every regular of TeX.SX knows, I like to use `expl3` code for solving problems, because I firmly believe in its advantages over traditional TeX and L^AT_EX programming.

This paper is mostly dedicated to “templating”, an idea I’m getting fond of. There is actually a “templating layer” for the future L^AT_EX3, but it’s not yet in polished shape. The Oxford Dictionary of English tells us that the primary meaning of *template* is “a shaped piece of rigid material used as a pattern for processes such as cutting out, shaping, or drilling”, more generally “something that serves as a model for others to copy” and this is how I think about *templating* here. This article is not about “document templates”, the (often unfortunately bad) pieces of code that users are sometimes forced to fill in.

Basically, the templating I’m going to describe is achieved by defining a temporary function (macro, if you prefer) to do some job during a loop. This has the advantage of allowing us to use the standard placeholders such as `#1`, `#2` and so on instead of macros, as is done (for instance) with the popular `\foreach` macro of TikZ/PGF, with concomitant expansion problems. For instance, with

```
\foreach \i in {1,2,3} { -\i- }
```

the loop code doesn’t “see” 1, 2 and 3, but `\i`, which *expands* to the current value. In `expl3` we have

```
\clist_map_inline:nn {1,2,3} { -#1- }
```

and this is possible by the same templating technique I’ll describe.

There are a couple of unrelated topics, just to show other applications of loops.

Technical note. The code shown here may differ slightly from the post on TeX.SX, but the functionality is the same. Sometimes I had afterthoughts,

¹ For the first installment, “TeX.StackExchange cherry picking: `expl3`”, see *TUGboat* 39:1, pp. 51–59, tug.org/TUGboat/tb39-1/tb121gregorio-expl3.pdf. Some terminology introduced there is used here without explanation.

or decided to use new functions added to `expl3` in the meantime. As in the first installment, the code I show should be thought of as surrounded by `\ExplSyntaxOn` and `\ExplSyntaxOff` (except for macro calls), and `\usepackage{xparse}` is mandatory.

2 Euclid’s algorithm

I’ve taught algebra classes for a long time and the Euclidean algorithm for the greatest common divisor has always been a favorite topic, because it’s a very good way to introduce several ideas, particularly recursion. The algorithm consists in repeating the same operations until we get to 0. Thus any implementation must clearly show recursion at work. Here’s the code for it.² (Well, a reduced version thereof for nonnegative numeric input only.)

```
\NewExpandableDocumentCommand{\euclid}{mm}
{
  \egreg_euclid:nn { #1 } { #2 }
}

\cs_new:Nn \egreg_euclid:nn
{
  \int_compare:nTF { #2 = 0 }
  { #1 } % end
  {
    \egreg_euclid:nf
    { #2 }
    { \int_mod:nn { #1 } { #2 } }
  }
}

\cs_generate_variant:Nn \egreg_euclid:nn { nf }
```

Some implementations do a swap if the first number is less than the second, but this is unnecessary, because the algorithm itself will perform it. The terminating condition is reached when we get a remainder of 0, in which case we output the first number. Otherwise the function calls itself with the first argument being the previous second one, and the second argument being the current remainder of the division.

Pure and simple: the only trick, for greater efficiency, is to call a variant of the main function in order to fully expand the mod operation.

It would be much more fun to implement the mod operation in fully expandable fashion, but somebody has already done it! Let’s enjoy laziness!

My answer also features a possibly interesting `\fulleuclid` macro that *prints* the steps.

3 Mappings

Data can be available in “serial” form; `expl3` has several data types of this kind: *clists* (comma separated lists), *sequences* and *property lists*. Further, token

² <https://tex.stackexchange.com/q/453877/>

lists can be seen as a list of tokens (or braced groups). For each of these data types `expl3` provides mapping functions that process each item of the given data.

The first toy problem consists in making a short table of contents exploiting `\nameref`.³ The user inputs something like

```
\procedurelist{
  PM-tyinglaces,
  PM-polishshoes,
  PM-ironshirt
}
```

where the input is a comma separated list of labels and the document contains something like

```
\section{Tying laces}\label{PM-tyinglaces}
```

for each label used. This should print a table with section numbers in the first column and section titles in the second column.

```
\NewDocumentCommand{\procedurelist}{m}
{
  \begin{tabular}{c1}
  \toprule
  \multicolumn{1}{c}{\textbf{PM}} & \textbf{Name} \\
  \midrule
  \clist_map_function:nN
  { #1 }
  \__kjc_procedurelist_row:n
  \bottomrule
  \end{tabular}
}
\cs_new:Nn \__kjc_procedurelist_row:n
{ \ref{#1} & \nameref{#1} \\ }
```

The `\clist_map_function:nN` command splits its first argument at commas, trimming spaces before and after each item and discarding empty items, and then passes each item to the function specified as the second argument. A variant is also available, `\clist_map_function:NN`, which expects as its first argument a *clist variable*.

In this case, the auxiliary function which the mapping is handed to is a template for a table row.

In contrast to other approaches, the operation is not hindered by being in a table, because the complete list of tokens

```
\__kjc_procedurelist_row:n {PM-tyinglaces}
\__kjc_procedurelist_row:n {PM-polishshoes}
\__kjc_procedurelist_row:n {PM-ironshirt}
```

is built before \TeX starts expanding the first macro.

A similar approach with

```
\clist_map_inline:nn
{ #1 }
{ \ref{##1} & \nameref{##1} \\ }
```

would not work here, because the mapping would start in a table cell and end in another one, which is impossible.

However, the user here is far-seeing and is worried about their boss not liking the appearance of

the table or perhaps wanting such tables printed in slightly different ways across the document.

We can make the auxiliary function variable! Here's the code.

```
\NewDocumentCommand{\procedurelist}{om}
{
  \group_begin:
  \IfValueT{#1}
  {
    \cs_set:Nn \__kjc_procedurelist_row:n { #1 }
  }
  \begin{tabular}{c1}
  \toprule
  \multicolumn{1}{c}{\textbf{PM}} & \textbf{Name} \\
  \midrule
  \clist_map_function:nN
  { #2 }
  \__kjc_procedurelist_row:n
  \bottomrule
  \end{tabular}
  \group_end:
}
\cs_new:Nn \__kjc_procedurelist_row:n
{ \ref{#1} & \nameref{#1} \\ }
```

We added an optional argument to the main macro. If this argument is present, then it will be used to redefine the auxiliary function. Grouping is used in order not to clobber the standard meaning of the auxiliary function. The same call as before would print the same table, but a call such as

```
\procedurelist[\ref{#1} & \textit{\nameref{#1}} \\]{
  PM-tyinglaces,
  PM-polishshoes,
  PM-ironshirt
}
```

would apply `\textit` to each section title. This is what I call *templating*.

4 Dummy variables

A user asked for a `\replaceproduct` macro so that `\replaceproduct{p_i^{\varepsilon_i}}{3}{n}` produces

$$p_1^{\varepsilon_1} p_3^{\varepsilon_3} p_3^{\varepsilon_3} \dots p_n^{\varepsilon_n}$$

The idea is that the “dummy variable” i is replaced by the numbers 1, 2 and 3 (the value is given in the second argument) and by n at the end (third argument).⁴

The idea of templating and using `#1` does not have the same appeal as before, because the call would need to be

```
\replaceproduct{p_#1^{\varepsilon_#1}}{3}{n}
```

which is more obscure. What if we make the dummy variable an optional argument, with default value `i`? Maybe we need to use `k` because the main template contains the imaginary unit or we want to be obscure at all costs (see the final example).

³ <https://tex.stackexchange.com/q/451423/>

⁴ <https://tex.stackexchange.com/q/448389/>

However, \TeX doesn't allow placeholders for parameters other than #1, #2 and so on—but we have `expl3` regular expression search and replace. Can we store the template in a token list variable and replace all appearances of the dummy variable with #1? Certainly we can: assuming that the template is the second argument in the macro we're going to define, we can do

```
\tl_set:Nn \l__egreg_rp_term_tl { #2 }
```

Now we can search and replace the dummy variable, which is given as (optional) argument #1, by

```
\regex_replace_all:nnN
{ #1 } % search
{ \cB\{ \cP\#1 \cE\} } % replace
\l__egreg_rp_term_tl % what to act on
```

so that each appearance of `i` (or whatever letter we specified in the optional argument) is changed into `{#1}`.⁵ The prefix `\cB` to `\{` is not standard in `regexes`⁶; in \TeX we need to be careful with category codes, so `\cB` says that the next token must be given the role of a “begin group” characters. Similarly for `\cE` as “end group” and `\cP` for “parameter”.

We now face the final problem: how to pass this to the auxiliary function that serves as internal template? We should do

```
\cs_set:Nn \__egreg_rp_term:n {(template)}
```

but the template is stored in a token list variable that we need to get the value of. That's a known problem and `expl3` already has the solution: the argument type `V`, that denotes take the *value* of the variable and pass it, braced, as an argument to the main function. So we just need to define a suitable variant of `\cs_set:Nn`, namely

```
\cs_generate_variant:Nn \cs_set:Nn { NV }
```

We now have all the ingredients and we can bake our cake:

```
\NewDocumentCommand{\replaceproduct}{0{i}mmm}
{#1 = item to substitute
 % #2 = main terms
 % #3 = first terms
 % #4 = last term
 \group_begin:
 \egreg_rp:nnnn { #1 } { #2 } { #3 } { #4 }
 \group_end:
}
\tl_new:N \l__egreg_rp_term_tl
\cs_generate_variant:Nn \cs_set:Nn { NV }
\cs_new:Nn \egreg_rp:nnnn
{
 \tl_set:Nn \l__egreg_rp_term_tl { #2 }
 \regex_replace_all:nnN
 { #1 } % search
 { \cB\{ \cP\#1 \cE\} } % replace
 \l__egreg_rp_term_tl % what to act on
```

⁵ \TeX nical remark: things are set up so that # tokens are not unnecessarily doubled when stored in a token list variable.

⁶ Informal abbreviation for “regular expressions”.

```
\cs_set:NV \__egreg_rp_term:n \l__egreg_rp_term_tl
\int_step_function:nN { #3 } \__egreg_rp_term:n
\cdots
\__egreg_rp_term:n { #4 }
}
```

The only new bit is `\int_step_function:nN`, which is similar to the mapping functions, but uses the most natural series: the numbers from 1 up to the integer specified as end; each number, in its explicit decimal representation, is passed to the auxiliary function as in the mapping of the previous section. In the example call, the third argument is 3, so there will be three steps.

The function `\int_step_function:?N` comes in three flavors:

```
\int_step_function:nN
\int_step_function:nnN
\int_step_function:nnnN
```

In the two-argument version we only specify the end, in the three-argument version we specify the start and end; in the full version the three `n`-type arguments specify starting point, step and final point. Thus the calls

```
\int_step_function:nN {3} \__egreg_rp_term:n
\int_step_function:nnN {1} {3} \__egreg_rp_term:n
\int_step_function:nnnN {1} {1} {3} \__egreg_rp_term:n
```

are equivalent.

We can call the macro like

```
\[ \replaceproduct{p_i^{\varepsilon_i}}{3}{m} =
\replaceproduct{j}{i_j^{\eta_j}}{2}{n} \]
```

to get

$$p_1^{\varepsilon_1} p_2^{\varepsilon_2} p_3^{\varepsilon_3} \dots p_m^{\varepsilon_m} = i_1^{\eta_1} i_2^{\eta_2} \dots i_n^{\eta_n}$$

The same idea can be used for a “macro factory” where the task is to define one-parameter macros using the keyword `PARAM` in place of #1.⁷ The problem here is that apparently plain \TeX is used, but it's not really difficult: `expl3` can also be used on top of it.

The input `\foo{qix}{: : PARAM : :}` should be equivalent to

```
\def\barqix#1{: : #1 : :}
```

and this can be accomplished easily.

```
\input expl3-generic
```

```
\ExplSyntaxOn
\tl_new:N \l__egreg_param_tl
```

```
\cs_new_protected:Npn \foo #1 #2
{
 \tl_set:Nn \l__egreg_param_tl { #2 }
 \regex_replace_all:nnN
 { PARAM }
 { \cP\#1 }
 \l__egreg_param_tl
 \cs_set:NV \__egreg_param:n \l__egreg_param_tl
```

⁷ <https://tex.stackexchange.com/q/355568/>

```

\cs_new_eq:cN {bar#1} \__egreg_param:n
}
\cs_generate_variant:Nn \cs_set:Nn { NV }
\ExplSyntaxOff

\foo{qix}{: : PARAM : :}

\barqix{hi world}

\bye

```

The function `__egreg_param:n` is temporary, but necessary: a variant such as `\cs_new:cpV` cannot be defined because of the parameter text which can consist of an arbitrary number of tokens to jump over. Here I exploit the fact that `\cs_set:Nn` computes its parameter text from the signature.

5 Double loops

We want to be able to generate and print a triangular diagram such as

```

1 × 1 = 1
2 × 1 = 2  2 × 2 = 4
3 × 1 = 3  3 × 2 = 6  3 × 3 = 9
4 × 1 = 4  4 × 2 = 8  4 × 3 = 12  4 × 4 = 16

```

with as little effort as possible.⁸

This calls for using `array`, but it also needs a double loop, which makes it inconvenient to use `\int_step_function:nN` as before, because only one argument is passed to the auxiliary function. The posted answer by Jean-François Burnol (jfbu) is as usual very nice, but not easily extendable — we may want only the operations, instead of also showing the result; or to do addition instead of multiplication; etc.

My idea is to define a macro `\lowertriangular` which takes as arguments the number of rows and what to do with the indices in each cell; for instance, the diagram above would be generated by

```
\lowertriangular{4}{#1\times #2 = \inteval{#1*#2}}
```

The macro `\inteval` is provided by `xfp`, which is part of the `xparse` family accompanying `expl3` and needs to be loaded. A lower triangular matrix can be generated by

```
\left[\lowertriangular{4}{a_{#1#2}}\right]
```

After all, I teach linear algebra courses, so triangular matrices are my bread and butter.

The placeholders `#1` and `#2` stand, respectively, for the row and column index.

We need nested loops, the outer one stepping the row index, the inner one stepping the column index, but only up to the row index. However, since we have to make an `array`, we need to build the body beforehand and then feed it to the matrix building environment. Small complication: the indices provided

by the outer loop are called `##1`, those relative to the inner loop are called `####1` (it’s quite predictable, but has to be mentioned).

As I said, we have to use the “inline” form for the loop, that is, `\int_step_inline:nn` (which has siblings like those described before for the similar `\int_step_function:nN`). When we are at the “cell level”, we will use the auxiliary function defined with the template given as second argument to the user level macro.

```

\NewDocumentCommand{\lowertriangular}{mm}
{
  \group_begin:
  \egreg_lt_main:nn { #1 } { #2 }
  \group_end:
}

\tl_new:N \l__egreg_lt_body_tl

\cs_new_protected:Nn \egreg_lt_main:nn
{
  % an auxiliary function for massaging the entries
  \cs_set:Nn \__egreg_lt_inner:nn { #2 }
  % clear the table body
  \tl_clear:N \l__egreg_lt_body_tl
  % outer loop, #1 rows
  \int_step_inline:nn { #1 }
  {
    % inner loop, ##1 columns
    \int_step_inline:nn { ##1 }
    {
      % add the entry for row ##1 (outer loop),
      % column ####1 (inner loop)
      \tl_put_right:Nn \l__egreg_lt_body_tl
        { \__egreg_lt_inner:nn { ##1 } { ####1 } }
      % if ##1 = ####1 end the row,
      % otherwise end the cell
      \tl_put_right:Nx \l__egreg_lt_body_tl
        {
          \int_compare:nTF { ##1 = ####1 }
            { \exp_not:N \ } % end row
            { & }          % end cell
        }
    }
  }
  % output the table
  \begin{array}{ @{} *{#1}{c} @{} }
  \l__egreg_lt_body_tl
  \end{array}
}

```

An almost straightforward modification of the code allows for producing upper as well as lower triangular matrices. It’s sufficient to add a test to make the inner loop go on all the way, instead of stopping at the diagonal.

```

\NewDocumentCommand{\lowertriangular}{mm}
{
  \group_begin:
  \egreg_tm_main:nnn { #1 } { #2 } { >= }
  \group_end:
}

\NewDocumentCommand{\uppertriangular}{mm}
{

```

⁸ <https://tex.stackexchange.com/q/435349/>

```

\group_begin:
\egreg_tm_main:nnn { #1 } { #2 } { <= }
\group_end:
}

\tl_new:N \l__egreg_tm_body_tl

\cs_new_protected:Nn \egreg_tm_main:nnn
{% #1 = size, #2 = template, #3 = < or >
% an auxiliary function for massaging the entries
\cs_set:Nn \__egreg_tm_inner:nn { #2 }
% clear the table body
\tl_clear:N \l__egreg_tm_body_tl
% outer loop, #1 rows
\int_step_inline:nn { #1 }
{
% inner loop, #1 columns
\int_step_inline:nn { #1 }
{
% add the entry for row ##1 (outer loop),
% column #####1 (inner loop) only if
% ##1 #3 #####1 is satisfied
\int_compare:nT { ##1 #3 #####1 }
{
\tl_put_right:Nn \l__egreg_tm_body_tl
{ \__egreg_tm_inner:nn { ##1 } { #####1 } }
}
% if #####1 = #1 end the row,
% otherwise end the cell
\tl_put_right:Nx \l__egreg_tm_body_tl
{
\int_compare:nTF { #####1 = #1 }
{ \exp_not:N \ } % end row
{ & } % end cell
}
}
}
}
% output the table
\begin{array}{@{} *{#1}{c} @{} }
\l__egreg_tm_body_tl
\end{array}
}

```

Now `\uppertriangular{5}{a_{#1#2}}` prints the body of an upper triangular matrix and my linear algebra course can go on. Particularly because I can also define

```

\NewDocumentCommand{\diagonal}{mm}
{
\group_begin:
\egreg_tm_main:nnn { #1 } { #2 } { = }
\group_end:
}

```

and get the diagonal matrices I need. I leave as an exercise the further extension of defining an optional template for filling the otherwise empty cells.

The integer comparison `\int_compare:n(TF)` accepts quite complex tests in its first argument, but here we're interested in what operators are allowed; they are `'= < > != <= >='` and their meaning should be obvious. In a perfect world they would be `'= < > ≠ ≤ ≥'`, but let's be patient.⁹

⁹ This could easily be added for Unicode engines such as XeTeX or LuaTeX; it would be more complicated to support

6 A templating puzzle

First let me present the code:¹⁰

```

\NewDocumentCommand{\automagic}{mm}
{
\begin{figure}
\clist_map_inline:nn { #1 }
{
\cs_set:Nn \__oleinik_automagic_temp:n
{
\caption { #2 }
}
\begin{subfigure}[t]{0.33\textwidth}
\includegraphics[
width=\textwidth,
]{example-image-##1}
\__oleinik_automagic_temp:n { #1 }
\end{subfigure}
}
\end{figure}
}

```

If one uses

```
\automagic{a,b,c}{Figure #1 from the set: ‘‘##1’’}
```

the result would show the three subcaptions

This is figure a from the set: “a,b,c”

This is figure b from the set: “a,b,c”

This is figure c from the set: “a,b,c”

The trick is that `\clist_map_inline:nn` does its own templating. The interested reader may enjoy solving the puzzle.

7 ISBN and ISSN

Every book has a number, called ISBN (International Standard Book Number) and each serial journal has an ISSN (International Standard Serial Number).

Originally, ISBN consisted of ten digits (with the final one being possibly X); later the code was extended to thirteen digits, but in a way that allowed old numbers to fit in the scheme by adding ‘978’ at the beginning and recomputing the final digit, which is a checksum. For instance, *The TeXbook* originally had ISBN 0201134489, while more recent editions have 9780201134483. After the leading 978 there is a 0, which means the book has been published in an English-speaking country. The rest denotes the publisher and the issue number internal to the publisher. Books published in Brazil will start with 97865 or 97885; books published in Italy with either 97888 or 97912. The 979 prefix is a more recent extension for coping with a greater number of books.

On the contrary, the eight digit ISSN doesn't convey information about the place of publication; it's basically a seven digit number with a final checksum (which can be X). Why this strange possibility?

legacy 8-bit engines and those symbols in every encoding that has them. Code portability is much more important.

¹⁰ <https://tex.stackexchange.com/q/410913/>

Because the checksum is computed modulo 11, so the remainder can be from 0 to 10 and X represents 10. This is also the case for old style ISBN, whereas the new codes compute the checksum modulo 10.

The algorithm for verifying correctness of an old ISBN is simple: the first digit is multiplied by 10, the second by 9 and so on up to the last digit (or X) which is multiplied by 1. All numbers are added and the result should be a multiple of 11. This method is guaranteed to catch errors due to transpositions of adjacent digits, but is not otherwise foolproof. For ISSN it is the same, but starting with multiplication by 8.

For a new style ISBN, the first digit is multiplied by 1, the second by 3, the third by 1 and so on, alternating 1 and 3. The sum of all numbers so obtained should be a multiple of 10 (no X needed).

We would like to have a macro for checking the validity of an ISBN or ISSN.¹¹ The package `ean13isbn` can be used for printing the bar code corresponding to a valid ISBN.

I provided a solution with \TeX arithmetic a while ago. Now it's time to do it in `expl3`. The numbers may be presented with various hyphens, for separating the relevant information, but this is neither recommended nor required. Thus the macros first remove all hyphens and act on the string of numerals that result.

Since the methods for computing checksums are very similar, we can dispense with much code duplication. I define two user level macros, `\checkISBN` and `\checkISSN`. Both first remove the hyphens and then check the lengths. If this test passes, control is handed to a function that has as arguments the length and the modulo (11 for ISSN and old style ISBN, 10 for new style ISBN). The multipliers are kept in constant sequences defined beforehand.

This function will set a temporary sequence equal to the one corresponding to the length, then computes the checksum and the remainder of the division with the prescribed modulo. If the remainder is 0, the code is deemed valid.

An important feature we exploit is that in the first and third arguments to `\int_compare:nNnTF` any *integer denotation* is allowed, with full expansion; so we can use our friend `\int_step_function:nN` to extract the multiplier and the digit, insert `*` between them (for multiplication) and add a trailing `+`. The final digit is treated specially, because it may be X; in this case 10 is used.

```
\NewDocumentCommand{\checkISBN}{m}
{
  \__egreg_check_normalize:Nn
```

```
  \l__egreg_check_str
  { #1 }
  % ISBN can have length 10 or 13
  \int_case:nnF { \str_count:N \l__egreg_check_str }
  {
    {10}{\__egreg_check:nn { 10 } { 11 }}
    {13}{\__egreg_check:nn { 13 } { 10 }}
  }
  {Invalid~(bad~length)}
}
\NewDocumentCommand{\checkISSN}{m}
{
  \__egreg_check_normalize:Nn
  \l__egreg_check_str
  { #1 }
  % ISSN must have length 8
  \int_compare:nNnTF
  { \str_count:N \l__egreg_check_str } = { 8 }
  { \__egreg_check:nn { 8 } { 11 } }
  {Invalid~(bad~length)}
}

\str_new:N \l__egreg_check_str

\seq_const_from_clist:cn {c_egreg_check_8_seq}
{ 8,7,6,5,4,3,2,1 }
\seq_const_from_clist:cn {c_egreg_check_10_seq}
{ 10,9,8,7,6,5,4,3,2,1 }
\seq_const_from_clist:cn {c_egreg_check_13_seq}
{ 1,3,1,3,1,3,1,3,1,3,1,3,1 }

% remove hyphens
\cs_new_protected:Nn \__egreg_check_normalize:Nn
{
  \str_set:Nn #1 { #2 }
  \str_replace_all:Nnn #1 { - } { }
}

% the main macro
\cs_new_protected:Nn \__egreg_check:nn
{ % #1 = length, #2 = modulo
  % use the appropriate constant sequence
  \seq_set_eq:Nc
  \l__egreg_check_seq
  { c_egreg_check_#1_seq }
  % compute the checksum and check it
  \int_compare:nNnTF
  {
    \int_mod:nn
    { \__egreg_check_aux_i:n { #1 } }
    { #2 }
  }
  = { 0 }
  {Valid}
  {Invalid~(bad~checksum)}
}
\cs_new:Nn \__egreg_check_aux_i:n
{ % do a loop from 1 to 7, 9 or 12
  \int_step_function:nN
  { #1-1 }
  \__egreg_check_aux_ii:n
  % and add the last digit
  \str_if_eq:eeTF
  { \str_item:Nn \l__egreg_check_str { #1 } }
  { X }
  { 10 }
  { \str_item:Nn \l__egreg_check_str { #1 } }
}
```

¹¹ <https://tex.stackexchange.com/q/39719/>

```
% the auxiliary function extracts the items from
% the sequence (multiplier) and the string (digit)
\cs_new:Nn \__egreg_check_aux_ii:n
{
  \seq_item:Nn \l__egreg_check_seq { #1 }
  *
  \str_item:Nn \l__egreg_check_str { #1 }
  +
}
```

Check with the following test:

```
\checkISBN{12345}           % invalid
\checkISBN{111111111X}     % invalid
\checkISSN{1234-56789}     % invalid
\checkISSN{1234-567X}     % invalid
\checkISBN{0201134489}     % TeXbook
\checkISBN{978-0201134483} % TeXbook
\checkISSN{0896-3207}     % TUGboat
```

With the same idea one could devise a fully expandable macro that takes as input a string of digits, applies a sequence of weights and computes a check digit based on a modulo operation.

8 Catcode tables

Every TeX user is fond of category codes, particularly when they put sticks in the wheels.¹² How to print on the terminal and log file the current status of category codes?¹³

We need a macro that calls a loop; with legacy TeX engines, the table is limited to the range 0–255, but with Unicode engines we can go much further. Another issue is that characters in the range 0–31 and 127–255 may fail to print in the log file, so I'll adopt for them the usual $\hat{\hat{char}}$ or $\hat{\hat{char}}\langle char \rangle$ convention. For instance, character 0 is represented by $\hat{\hat{0}}$, character 127 by $\hat{\hat{?}}$, but character 128 by $\hat{\hat{80}}$.

The macro can be called like `\catcodetable`, `\catcodetable[255]` or `\catcodetable[0-255]` all meaning the same thing: no optional argument implies 0–255; a single number specifies the end point, starting from 0; two numbers separated by a hyphen specify start and end points.

I use an `\int_step_function:nnnN` loop, the auxiliary function prints the code point (in decimal), then a representation of the character, then its category code in verbose mode. The interesting bit here, besides the complex tests for `\int_compare:nTF`, is `\char_generate:nn`. This function takes as arguments two numeric expressions; the first one denotes the code point, the second one the category code to assign. Of course only some of these catcodes are meaningful: 9, 14 and 15 aren't; also 13 cannot (yet) be used with XeTeX. I use here 12, for safety.

¹² In Italian we say *mettere i bastoni fra le ruote* when somebody tries to impede our endeavor.

¹³ <https://tex.stackexchange.com/q/60951/>

```
\NewDocumentCommand{\catcodetable}
{
  >\SplitArgument{1}{-}0{0-255}
}
{
  \catcodetablerange#1
}
\NewDocumentCommand{\catcodetablerange}{mm}
{
  \IfNoValueTF{#2}
  {
    \egreg_cctab:nn { 0 } { #1 }
  }
  {
    \egreg_cctab:nn { #1 } { #2 }
  }
}
\str_const:Nn \c_egreg_cctab_prefix_str { ^ ^ }
\cs_new_protected:Nn \egreg_cctab:nn
{
  \int_step_function:nnnN
  { #1 } % start
  { 1 } % step
  { #2 } % end
  \egreg_cctab_char:n
}
\cs_new_protected:Nn \egreg_cctab_char:n
{
  \iow_term:x
  {
    Code~\int_to_arabic:n { #1 } :~(
    \int_compare:nTF { 0 <= #1 < 32 }
    {
      \c_egreg_cctab_prefix_str
      \char_generate:nn { #1+64 } { 12 }
    }
    {
      \int_compare:nTF { #1 = 127 }
      {
        \c_egreg_cctab_prefix_str
        \char_generate:nn { #1-64 } { 12 }
      }
      {
        \int_compare:nTF { 128 <= #1 < 256 }
        {
          \c_egreg_cctab_prefix_str
          \int_to_hex:n { #1 }
        }
        {
          \char_generate:nn { #1 } { 12 }
        }
      }
    }
  )~\__egreg_cctab_catcode:n { #1 }
}
}
\cs_new:Nn \__egreg_cctab_catcode:n
{
  \int_case:nn { \char_value_catcode:n { #1 } }
  {
    {0}{escape}
    {1}{begin~group}
    {2}{end~group}
    {3}{math~shift}
    {4}{alignment}
  }
```

```

{5}{end~of~line}
{6}{parameter}
{7}{superscript}
{8}{subscript}
{9}{ignored}
{10}{space}
{11}{letter}
{12}{other~character}
{13}{active~character}
{14}{comment}
{15}{ignored}
}
}

```

A selected part of the output from `\catcodetable` with 8-bit \LaTeX :

```

Code 0: (^@) ignored
Code 1: (^A) active character
Code 2: (^B) active character
[...]
Code 31: (^_) active character
Code 32: ( ) space
Code 33: (!) other character
Code 34: (") other character
Code 35: (#) parameter
Code 36: ($) math shift
Code 37: (%) comment
Code 38: (&) alignment
Code 39: (') other character
[...]
Code 63: (?) other character
Code 64: (@) other character
Code 65: (A) letter
Code 66: (B) letter
[...]
Code 90: (Z) letter
Code 91: ([) other character
Code 92: (\) escape
Code 93: (]) other character
Code 94: (^) superscript
Code 95: (_) subscript
Code 96: (') other character
Code 97: (a) letter
Code 98: (b) letter
Code 122: (z) letter
Code 123: ({) begin group
Code 124: (|) other character
Code 125: (}) end group
Code 126: (~) active character
Code 127: (^?) ignored
Code 128: (^80) active character
Code 129: (^81) active character
[...]

```

Running `\catcodetable["10FFFF]` with \XeTeX also works, and produces a 37 MiB¹⁴ log file ending with

```

Code 1114109: (<U+10FFFD>) other character
Code 1114110: (<U+10FFFE>) other character
Code 1114111: (<U+10FFFF>) other character
)
Here is how much of TeX's memory you used:
9287 strings out of 492956
183011 string characters out of 6133502
204291 words of memory out of 5000000
[...]

```

The `<U+10FFFF>` is an artifact of `less` on my system.

A small curiosity about the code for the string constant that prints the two carets when needed:

```
\str_const:Nn \c_ereg_cctab_prefix_str { ^ ^ }
```

There *must* be a space between the carets, otherwise the standard \TeX convention would prevail and the string would end up containing character $32+64 = 96$, that is, `'`. The (ignored) space in between the carets separates them and so we get our desired two carets in the output string.

Happy \LaTeX ing!

◇ Enrico Gregorio
Dipartimento di Informatica
Università di Verona
and
 \LaTeX Team
enrico.gregorio@univr.it

¹⁴ "mebibyte"; 1 MiB = 2^{20} bytes.

Real number calculations in L^AT_EX: Packages

Joseph Wright

1 Background

T_EX does not include any “native” support for floating point calculations, but that has not stopped lots of (L^A)T_EX users from wanting to do sums (and more complicated things) in their document. As T_EX is Turing complete, it’s not a surprise that there are several ways to implement calculations. For end users, the differences between these methods are not important: what is key is what to use. Here, I’ll give a bit of background, look at the various possibilities, then move on to give a recommendation.

2 History

When Knuth wrote T_EX, he had one aim in mind: high-quality typesetting. He also wanted to have sources which were truly portable between different systems. At the time, there was no standard for specifying how floating point operations should be handled at the hardware level: as such, no floating point operations were system-independent.

Thus, Knuth decided that T_EX would provide no user access to anything dependent on platform-specific floating-point operations, and not rely on them within algorithms that produce typeset output. That means that the T_EX variables and operations that look like numeric floats (in particular dimensions) actually use *integer* arithmetic and convert “at the last minute”.

3 Technical considerations

There are two basic approaches to setting up floating point systems in T_EX: either using dimensions or doing everything in integer arithmetic.

Using dimensions, the input range is limited and the output has restricted accuracy. But on the other hand, many calculations are quite short and they are fast. On the other hand, if everything is coded in integer arithmetic, the programmer can control the accuracy completely, at the cost of speed.

Although it’s not an absolute requirement, ϵ -T_EX does make doing things a bit easier: rather than having to shuffle everything a piece at a time, it is possible to use in-line expressions for quite a lot of the work.

Another key technical aspect is expandability. This is useful for some aspects of T_EX work, particularly anywhere that it “expects a number”: only expansion is allowed in such places.

Another thing to consider is handling of T_EX registers as numbers. Converting for example a length

into something that can be used in a floating point calculation is handy, and it matches what ϵ -T_EX does for example in `\numexpr`. But in macro code it has to be programmed in.

The other thing to think about here is functionality: what is and isn’t needed in terms of mathematics. Doing straightforward arithmetic is clearly easier than working out trigonometry, logarithms, etc. What exactly you need will depend on the use case, but in principle, more functionality is always better.

4 (Package) options

For simple work using the `dimen` approach is convenient and fast: it takes only a small amount of work to set up stripping off the `pt` part. I’m writing here for people who don’t want to delve into T_EX innards, so let’s assume a pre-packaged solution is what is required.

There are lots of possible solutions on CTAN which cover some or all of the above. I don’t want to get into a “big list”, so I’ll simply note here that the following are available on CTAN:

- `apnum`
- `calculator`
- `fltpoint`
- `pst-fp`
- `minifp`
- `realcalc`
- `xint`

Some of these have variable or arbitrary precision, while others work to a pre-determined level. They also vary in terms of functions covered, expandability and so on.

I want to focus in on three possible “contenders”: `fp`, `pgf` and the L^AT_EX3 FPU (part of `expl3`). All of these are well-known and widely-used, offer a full set of functions, and a form of expressions.

The `fp` package formally uses *fixed* not *floating* point code, but the key for us here is that it allows a wide range of high-precision calculations. It’s also been around for a *long* time. However, it’s quite slow and doesn’t have convenient expression parsing — it requires reverse Polish.

On the flip side, the arithmetic engine in `pgf` uses `dimens` internally, so it is (relatively) fast but is limited in accuracy. The range limits also show up in some unexpected places, as a lot of range reduction is needed to make everything work. On the other hand, `\pgfmathparse` does read “normal” arithmetic expressions, so it’s pretty easy to use. I’ll also come to another aspect below: there is a “swappable” floating point unit to replace the faster `dimen`-based code.

The L^AT_EX3 FPU is part of `expl3`, but is available nowadays as a document-level package `xfp`. In contrast to both `fp` and the `pgf` approach, the L^AT_EX3 FPU is expandable. Like `pgf`, using the FPU means we can use expressions, and we also get reasonable performance (Bruno Le Floch worked hard on this aspect). The other thing to note is that the FPU is intended to match behaviour specified in the decimal IEEE 754 standard, and that the team have a set of tests to try to make sure things work as expected.

There’s one other option that one must consider: Lua. If you can accept using only LuaT_EX, you can happily break out into Lua and use its ability to use the “real” floating point capabilities of a modern PC. The one wrinkle is that without a bit of work, the Lua code *doesn’t* know about T_EX material: registers and so on need to be pre-processed. It also goes without saying that using Lua means being tied to LuaT_EX!

5 Performance

To test the performance of these options, I’m going to use the L^AT_EX3 benchmarking package `l3benchmark`. I’m using a basic set up:

```
\usepackage{l3benchmark}
\ExplSyntaxOn
\cs_new_eq:NN
  \benchmark
  \benchmark:n
\ExplSyntaxOff
\newsavebox{\testbox}
\newcommand{\fptest}[1]{%
  \benchmark{\sbox{\testbox}{#1}}%
}
```

(The benchmarking runs perform the same step multiple times, so keeping material in a box helps avoid any overhead for the typesetting step.)

The command `\fptest{...}` was then used with the appropriate input, such as the `\fpeval` command below, to calculate a test expression using a range of packages.

As a test, I’m using the expression

$$\sqrt{2} \sin\left(\frac{40}{180}\pi\right)$$

or the equivalent. Using that, it’s immediately apparent that the `fp` package is by far the slowest approach (Table 1). Unsurprisingly, using Lua is the fastest by an order of magnitude at least. In the middle ground, the standard approach in `pgf` is fastest, but not by a great deal over using the L^AT_EX3 or `pgf` FPUs.

Table 1: Benchmarking results (LuaT_EX v1.07, T_EX Live 2018, Windows 10, Intel i5-7200)

Package	Time/10 ⁻⁴ s
<code>fp</code>	99.4
<code>pgf</code>	2.95
<code>pgf/xfp</code>	5.51
L ^A T _E X3 FPU	6.42
LuaT _E X	0.57

6 Recommendation

As you can see above, there are several options. However, for end users wanting to do calculations in documents I think there is a clear best choice: the L^AT_EX3 FPU.

```
\documentclass{article}
\usepackage{xfp}
\begin{document}
\fpeval{round(sqrt(2) * sind(40),2)}
\end{document}
```

(The test calculation uses angles in degrees, so where provided, a version of the sine function taking degrees as input was used: this is expressed in the L^AT_EX3 FPU as `sind`. The second argument to `round`, 2, is the number of places to which to round the result.)

You’d probably expect me to recommend the L^AT_EX3 package: I am on the L^AT_EX team. But that’s not the reason. Instead, it’s that the FPU is almost as fast as using `dimens` (see `pgf` benchmarking), but offers the same accuracy as a typical GUI application for maths. It also integrates into normal L^AT_EX conventions with no user effort. So it offers by far the best balance of features, integration and performance for “typical” users.

◇ Joseph Wright
Northampton, United Kingdom
`joseph dot wright (at)`
`morningstar2.co.uk`

Real number calculations in \TeX : Implementations and performance

Joseph Wright

1 Introduction

\TeX only exposes integer-based mathematics in any user-accessible functions. However, as \TeX allows a full range of programming to be undertaken, it is unsurprising that a number of authors have created support for floating or fixed point calculations. I recently looked at this area from an “end user” point of view (“Real number calculations in \LaTeX : Packages”, pages 69–70 in this issue), focusing on a small number of widely-used options. In this article, I will look at the full set of packages available, examining their performance and considering some of the challenges the implementations face.

2 Floating *versus* fixed point

The packages I’ll examine here cover both *floating* and *fixed* point work. In a floating point approach, the total number of digits doesn’t vary, but an exponent is used to scale the meaning: what we’d normally think of as “scientific notation”. In contrast, fixed point calculations mean exactly that: there are a set number of digits used, and they are never scaled. That means that in a fixed point approach, the number of digits in both the integer and decimal parts is set by the implementation.

Both approaches have advantages. Floating point working means that we gain a greater input range, but have to track and allow for the different meaning of the integer/decimal boundary. Fixed point code does not have to deal with the latter issue, and can be hard-coded for the known limits of values. However, the range is necessarily more limited, both in terms of the maximum value accepted and handling of very small values (where the fixed lower limit cuts off precision).

Finally, there are packages which offer arbitrary precision: code which works to a pre-determined number of places, but where that limit can be adjusted by the user. Arbitrary precision may be used with either fixed or floating point representation of the mantissa.

3 Precision, accuracy and input range

Two key questions can be asked about any floating point unit (FPU): how many places (or digits) does it calculate, and how accurate are those digits? In terms of the precision provided, we also have to consider that there are different aims: for example, both `apnum` and `xint` implement arbitrary precision.

Table 1: Precision targets

	<i>Approach</i>	<i>Precision</i> ^a
<code>apnum</code>	Arbitrary	20 ^b
<code>xint</code>	Arbitrary	20
<code>fp</code>	Fixed	16
<code>pst-fp</code>	Fixed	16
<code>xfp</code>	Floating	15
Lua	Floating	15
<code>minifp</code>	Fixed	8
<code>pgf/fpu</code>	Floating	7
<code>pgf</code>	Fixed	5
<code>calculator</code>	Fixed	5
<code>fltpoint</code>	Floating	5 ^c

^aPlaces in the decimal part; ^bMinimum precision;

^cApplies only to division; other operations may provide more digits

Comparing the *accuracy* of these different approaches is non-trivial: is getting 5-digit accuracy from a 5-digit fixed-point system “better” than getting 12-digit accuracy from a 15-digit floating-point approach? There is also the question of exactly which tests one picks: depending on the exact values chosen, different implementations may “win”. As such, I will not tabulate “accuracy” results, but rather comment where the user might wish to be cautious.

In terms of the *precision* of different packages, the exact approach depends on the package: for example, packages offering arbitrary precision have only a default precision. Table 1 summarises the various packages on CTAN and provided in current \TeX systems that work in this area. I have included one non-macro approach: using Lua in `LuaTeX`. Clearly this is viable for only a subset of users, but as we will see, it is an important option.

For `pgf`, I will consider two approaches: its native mathematical engine and the optional floating point unit. There is also a loadable option to use Lua for the “back end” of calculations: unsurprisingly, it performs in a very similar way to the direct use of Lua. (The number of decimal places returned stays compatible with the standard `pgf` approach: five places.) The `pgf` package also offers fast programming interfaces to all operations which require that each numerical argument be passed directly to a \TeX dimension register: I have not considered those here, but they do of course offer increased performance.

4 Implementation approaches

4.1 Overview

\TeX provides very limited support for calculations. In Knuth’s \TeX , we have the `\count` and `\dimen`

registers for storage, and the operations `\advance`, `\multiply` and `\divide` (the latter truncating rather than rounding). The ε -TeX extensions add expandable expression evaluation with the same fundamental abilities: `\numexpr` and `\dimexpr` (the one wrinkle being that division rounds). The `\numexpr` and `\dimexpr` primitives have an internal range greater than `\maxdimen`, and so for example $A \times B/C$ can be calculated even if $A \times B$ would overflow.

4.2 Dimensions *versus* integers

At the macro level, these primitives allow two basic approaches, either using integer-based calculations or using dimension-based ones. As you may already know, dimensions in TeX are actually (binary) fixed-point numbers: they are stored in `sp` (scaled points), but displayed in `pt`.

Approaches using dimensions are limited by the underlying TeX mechanisms: five decimal places and an upper limit of `\maxdimen` (16 383.999 98 pt). On the other hand, the basic operations are both easy to set up and fast. Other than the need to remove a trailing `pt`, arithmetic does not even require any macros.

So, it is possible to use dimensions as the underlying data store and to provide additional functionality on top of this. However, it is also worth nothing that the rather limited range of dimensions means that moderately large and small values must be scaled as a first step. This is a potential source of inaccuracy or range issues.

Using an integer-based approach, storage and calculation necessarily require a range of macros or `\count` registers. On the other hand, this approach leaves the programmer in complete control of the precision used. Integer and decimal parts of a number, plus potentially an exponent, can be held in separate registers or extracted from a suitable macro before arithmetic takes place.

4.3 Beyond arithmetic

Once one looks beyond simple arithmetic, issues such as range reduction and handling of transcendental functions become important. This is particularly true for internal workings. For example, the normal approach to calculating sines is to use Taylor series. As several terms are required, rounding errors in each term may accumulate significant inaccuracy. Thus it is typically the case that internal steps for these operations have to work at higher precision than the user-accessible results.

Range reduction requires careful handling to avoid introduction of systematic errors. This again leads to concern over internal precision, as for ex-

ample the number of places of π used internally can have a large impact on the final values produced.

4.4 Standards

In TeX macros, it makes sense to store numbers in decimal form. That contrasts with most floating point implementations, where underlying storage is binary. Both of these cases are covered by the IEEE754 standard, which is the primary reference for implementers of floating point units in both software and hardware.

The IEEE standard specifies not a single approach but a number of related ideas to do with data storage, handling of accuracy, dealing with exceptions and so on. Whilst most TeX implementations do not directly aim to implement a full IEEE754-compliant approach, the standard does give us a framework with which to compare aspects of behaviour.

One small wrinkle is that storing values in binary means that some exact decimals cannot be expressed. This shows up when using Lua for mathematics, for example

```
\directlua{tex.print(12.7 - 20 + 7.3)}
```

gives

```
-8.8817841970013e-16
```

rather than 0.

4.5 Expandability

When programming in TeX, the possibility of making code expandable is almost always a consideration. Expandable code for calculations can be used in a wider range of contexts than non-expandable approaches. Of course, one can always arrange to execute code before an expansion context; here's an example with the `fp` package:

```
\FPadd\result{1.234}{5.678}
```

```
\message{Answer is \result}
```

However, for the user, code which works purely by expansion means they do not have to worry about such issues. Implementing calculations “expandably” means that registers cannot be used. With ε -TeX, this is not a major issue as simple expressions can be used instead. Creating expandable routines for calculations is thus possible provided the underlying operation is itself expandable. A key example where that is not the case is measuring the width of typeset material, for example

```
\pgfmathparse{width("some text")}
```

Expandable implementations require that the programmer work hard to hold all results on the input stack. Achieving this without a performance

impact is a significant achievement. However, in and of itself this is not the most important consideration for choosing a solution.

5 Expressions

By far the simplest approach to handling calculations is to have one macro per operation, for example `\FPadd` (from the `fp` package) for adding two numbers. At a programming level this is convenient, but for users, expressions are much more natural. Several of the packages examined here offer expressions, either in addition to operation-based macros or as the primary interface.

Each package inherently defines its own syntax for such expressions. However, the majority use a simple format which one might regard as ASCII-math, for example

```
1.23 * sqrt(2) + sin(2.3) / exp(3)
```

to represent

$$1.23 \times \sqrt{2} + \frac{\sin 2.3}{e^3}$$

Expressions may also need to cope with scientific notation for numbers: this is most obvious when using a dimension-based “back-end”, as the values cannot be read directly.

Several of the packages considered here offer expression parsing: `fp` is notable in using a stack approach rather than the more typical inline expressions as shown above. However, as parsing itself has a performance impact, the availability of faster “direct” calculation macros is often a benefit.

6 Performance infrastructure

To assess the performance of the various options, I wrote a script which uses `l3benchmark` to run a range of operations for all of the packages covered here. This has the advantage of carrying out a number of runs to get a measurable time. All of these tests were run in a single `.tex` source, using LuaTeX 1.07 (TeX Live 2018) on an Intel i5-7200 running Windows 10.

The full test file (over 600 lines long!) is available from my website: texdev.net/uploads/2019/01/14/FPU-performance.tex. Comparison values for calculations were generated using Wolfram Alpha (wolframalpha.com) at a precision exceeding any of the methods used here.

7 Arithmetic

Basic arithmetic is offered by all of the packages considered here. I chose to test this using the combination of two constants

$$a = 1.2345432123454321$$

$$b = 6.7890987678909876.$$

Table 2: Basic operations, ordered by addition results

	Time/ 10^{-4} s			
	$a + b$	$a - b$	$a \times b$	a/b
<code>calculator</code>	0.03	0.03	0.02	0.88
<code>Lua</code>	0.16	0.18	0.17	0.17
<code>minifp</code>	0.29	0.26	0.77	2.20
<code>pgf</code>	0.78	0.75	0.74	1.32
<code>apnum</code>	0.94	0.95	2.55	3.15
<code>xfp</code>	1.44	1.40	1.79	1.97
<code>pst-fp</code>	3.35	3.25	5.51	22.60
<code>xint</code>	3.92	3.75	2.95	6.77
<code>fp</code>	4.52	4.25	14.50	23.80
<code>fltpoint</code>	5.92	12.30	200	123
<code>pgf/fpu</code>	6.48	6.18	6.07	7.18

Table 3: Trigonometry, ordered by sin results

	$\sin \theta$	$\sin^{-1} c$	$\operatorname{sind} \phi$	$\operatorname{sind}^{-1} c$	$\tan \psi$
<code>Lua</code>	0.19	0.19	0.19	0.22	0.19
<code>pgf</code>	0.49	0.37	0.33	0.36	—
<code>calculator</code>	2.74	13.30	3.69	—	—
<code>pgf/fpu</code>	5.24	3.40	4.28	3.52	—
<code>xfp</code>	5.65	14.60	4.22	16	7.77
<code>fp</code>	18.50	25	—	—	31.90
<code>apnum</code>	50.70	131	—	—	73.70
<code>minifp</code>	—	—	8.23	—	—

As is shown in Table 2, these operations take times in the order of microseconds to milliseconds.

As one would likely anticipate, using Lua (which can access the hardware FPU) is extremely fast for operations across the range and provides accurate results in all cases. Even faster, except for division, is `calculator`, which uses a very thin wrapper around `\dimen` register working. Performance across the other implementations is much more varied, with the high-precision expandable `xfp` out-performing many of the less precise packages, and working close to, for example, `pgf` even though the latter takes advantages of `\dimen` registers. At the slowest extreme, both `fp` and in particular `fltpoint` take significantly longer than other packages.

Accuracy is uniformly good for addition and subtraction, with rounding in the last place posing an issue in only two cases (`minifp` and `pgf`'s FPU approach). For multiplication and division, most implementations are accurate for all returned digits. Again, only issues with rounding at the last digit of precision (`pgf`) prevent a “full house” of accurate results.

8 Trigonometry

Calculation of sines, cosines, etc., represents a more significant challenge to a macro-based implementation than basic arithmetic. As outlined above, considerations such as internal accuracy and range reduction come into play, and performance can become very limited. Not all packages even attempt to work in this area, and `fltpoint`, `pst-fp` and `xint` all lack any support for such functions.

To test performance in trigonometric calculations, I have again picked a small set of constants for use in various equations:

$$\begin{aligned}c &= 0.1234567890 \\ \theta &= 1.2345432123454321 \\ \phi &= 56.123456 \\ \psi &= 8958937768937\end{aligned}$$

A consideration to bear in mind when dealing with trigonometry is the units of angles. Depending on the focus of the implementation, `sin` may expect an angle in either radians or degrees. It is of course always possible to convert from one to the other using simple arithmetic, and thus all packages offering trigonometric functions can be used with either unit. However, this may lead to artefacts due to range reduction and the precision of conversion. As such, I have only tabulated data for “native” functions: sine in degrees is referred to as `sind`; the data are summarised in Table 3.

Lua is again by far the fastest approach and is accurate for all of the sine calculations. (I have allowed the use of conversion between degrees and radian here: in contrast to macro-based approaches, this seems reasonable with a “real” programming language and hardware-level FPU support.)

The difference between `calculator` and `pgf` is notable, as both use `\dimen` registers behind the scenes: `pgf` is significantly faster. In accuracy terms, both `calculator` and `pgf` provide four decimal place accuracy, so this is not a question of trading accuracy for performance. Enabling the FPU for `pgf` here does not improve accuracy, but does cause a significant performance hit.

Looking at the more precise approaches, `xfp` is best in performing for calculation of sine, though it is much less impressive for inverse sine. Accuracy for sine is uniformly good, with `fp` failing at 17 digits, and the other packages correct for the full set of digits returned.

The calculation of $\tan \psi$ is included to emphasise the challenge of range reduction. The input is out-of-range for a number of packages which otherwise can calculate tangents. Only `fp` and `xfp` give the correct

result: both `apnum` and Lua give entirely erroneous results. The failure of Lua is perhaps surprising, but likely arises due to the IEEE754 specification for binary storage.

9 Other operations

There are plenty of other operations which we might wish to execute using a calculation package. As for trigonometry, I have only included operations with “native” support in Table 4. Coverage of these various operations is somewhat variable. For example, a^x may be supported only for integer powers, or may also be provided for non-integer powers. Similarly, pseudo-random number generation is not always implemented.

The `pgf` approach is once again fast for a range of operations, but does suffer in terms of accuracy: only three decimal places for \sqrt{a} , $\exp a$ and $\ln a$, and only one decimal place for a^b . This remains the case when enabling the `pgf` FPU. The `calculator` package also suffers from some loss of accuracy, and is correct to only three decimal places for \sqrt{a} .

Other implementations are generally successful in offering good accuracy: `minifp` to at least 7 places in all cases, and all other approaches to at least 14 places. As such, performance is once again the main difference between the various implementations, although the nature of available operations is also worth considering. Perhaps the most notable case is that whilst a^n is widely implemented, a^x is less well supported.

Generation of pseudo-random values is something of a special case. Modern \TeX engines offer primitive support for generation of such numbers, all using code originally written by Knuth for `META-FONT`. This is exploited by both `xfp` and `xint` to generate such numbers rapidly and expandably. In contrast, other implementations generate values purely in macros, and so are non-expandable (due to the need to track the seed between executions).

10 Conclusions

Implementing fully-fledged floating point support in \TeX is a significant programming challenge. It is also a challenge that has been solved by a number of talented \TeX programmers. There are several packages which offer good-to-excellent accuracy with precision of at least 8 places, and in some senses this means that choosing a package is complicated. However, unless one requires arbitrary precision, the balance of performance and precision is best managed by `xfp`, the \LaTeX 3 FPU as a user package. Whilst not the fastest for every single operation, it performs well across the board and offers performance often

Table 4: Extended operations, ordered by \sqrt{a} results

	\sqrt{a}	$\exp a$	$\ln a$	a^5	a^b	$\text{round}(d, 2)$	rand
Lua	0.18	0.19	0.19	0.16	0.17	0.20	0.15
pgf	0.84	0.67	0.54	0.60	1.28	—	0.07
xfp	3.88	7.43	10.20	15.50	14.80	4.11	1.47
minifp	4.15	8.96	13.10	3.61	—	0.46	2.18
pgf/fpu	4.96	4.50	4.04	7.51	8.60	—	1.12
calculator	6.91	9.21	29.10	0.32	38.90	1.28	—
xint	11.50	—	—	8.01	—	1.21	1.29
apnum	42.60	121	133	13.10	—	0.08	—
fp	79.70	20.20	40.80	66.20	67.90	—	19.90
fltpoint	—	—	—	—	—	1.61	—

comparable to approaches using $\text{T}_{\text{E}}\text{X}$ dimensions (which are thus restricted to only 5 decimal places at best). Where code is known to be strictly $\text{LuaT}_{\text{E}}\text{X}$ -only, using Lua is of course the logical choice: no macro implementation can compete with support at the binary level.

For arbitrary precision work, `apnum` is not only the best choice but also the only candidate if one wishes to use transcendental functions.

11 Acknowledgements

Thanks to all of the package authors who gave me feedback on my tests for their packages:

- Petr Olšák (`apnum`)
- Robert Fuster (`calculator`)
- Eckhart Guthöhrlein (`fltpoint`)
- Michael Mehlich (`fp`)
- Dan Luecking (`minifp`)
- Christian Feuersänger (`pgf`)
- Jean-François Burnol (`xint`)

Thanks also to $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ team members Bruno Le Floch, Frank Mittelbach, David Carlisle and Ulrike Fischer for suggestions on benchmarking the various packages. Bruno also implemented the $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}3$ FPU, and his efforts in making this both expandable and (relatively) fast are truly astounding.

◇ Joseph Wright
Northampton, United Kingdom
joseph dot wright (at)
morningstar2.co.uk

TeX4ht: L^AT_EX to Web publishing

Michal Hoftich

Abstract

The article gives overview of the current state of development of TeX4ht, a conversion system for (L^A)TeX to HTML, XML, and more. It introduces `make4ht`, a build system for TeX4ht as well as basic ways how to configure TeX4ht.

1 Overview of the conversion process

TeX4ht is a system for conversion of TeX documents to various output formats, most notably HTML and *OpenDocument Format*, supported by word processors such as Microsoft Word or LibreOffice Writer. An overview of the system is depicted in figure 1.

The package `tex4ht.sty` starts the conversion process. The document preamble is loaded as usual, but it keeps track of all loaded files. It loads special configuration files for any packages used that are supported by TeX4ht at the beginning of the document. These configuration files are named as the configured file with extension `.4ht`. They may fix clashes between the configured package and TeX4ht, but most notably the package commands are patched to insert special marks to the DVI file, so-called hooks.

After the package configuration, another type of `.4ht` files are loaded. These populate inserted hooks with tags in the selected output format. In the last step before processing of the document contents, a `.cfg` provided by the user can configure the hooks with custom tags. Compilation of the document then continues as usual, resulting in a special DVI file.

The generated DVI file is then processed with the `tex4ht` command. This command creates output files, converts input encodings to UTF-8, and creates two auxiliary files: an `.idv` file, a special DVI file that contains pages to be converted to images, which can be the contents of the L^AT_EX picture environment or complex mathematics; second, an `.lg` file with a list of output files, CSS instructions, and instructions for compiling individual pages in the `.idv` file to images.

The last step in the compilation chain is the `t4ht` program. It processes the `.lg` file and extracts the CSS instructions, converts the images in the `.idv` file, and may call various external commands.

2 Supporting scripts

Because the entire conversion process consists of several consecutive steps, we use scripts to make this

Translation by the author from the original in *Zpravodaj* 2018/1–4, pp. 11–21, for the BachTeX 2019 proceedings.

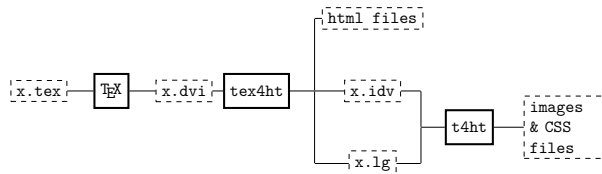


Figure 1: TeX4ht process overview

process easier. The TeX4ht distribution contains several such scripts. They differ in the supported output format, TeX engine used, and options passed to the underlying commands. The most commonly used script is `htlatex`, which uses the PDF_{TeX} engine with L^AT_EX and produces HTML output.

Each of the scripts loads the TeX4ht package without needing to specify it in the document, and options from the command line are passed to the package as well to `tex4ht` and `t4ht` commands.

For example, the following command can be used to request the output in XHTML format in the UTF-8 encoding:

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

However, these scripts are not flexible; each time, they execute a three-time compilation of the TeX document. This ensures the correct structure of hypertext links and tables, as they require multiple compilations to function properly, processing the document repeatedly with `tex4ht` and `t4ht`.

For example, if the document contains a bibliography or glossary that is created by external programs, it is necessary to first call `htlatex`, then the desired program, and then `htlatex` again. In the case of larger documents, compilation time may thus be relatively long.

Passing options to the underlying commands is also quite difficult.

New build scripts have been created for these reasons. My first project that attempted to simplify the TeX4ht compilation process was `tex4ebook`. It added support for e-books, specifically *ePub*, *ePub3* and *mobi* formats. It added support for use of command line switches and build files written in Lua.

The main difference between `tex4ebook` and TeX4ht is the third compilation step. The `t4ht` command is used only to create a CSS file. Image conversion and execution of the external commands is controlled by `tex4ebook` itself. In addition, thanks to the build file support, it is possible to execute commands between individual TeX compilations, for example, to execute an index processor or `bibtex` after the first compilation.

The library that added the build support provided features useful also for other output formats than e-books. It was extracted as a standalone tool and the `make4ht` build system is now a recommended tool for use of $\text{\TeX}4\text{ht}$.

3 The `make4ht` build system

`make4ht` enables creation of build scripts in the Lua language. It supports execution of arbitrary commands during the conversion, post-processing of the generated files and defining commands for image conversion. Using the so-called modes, it is possible to influence the order of compilation using switches directly from the command line. For example, the basic script used by `make4ht` supports a draft mode which runs only one compilation of the document instead of the usual three. This can be used to significantly speed up the compilation.

Currently, only \LaTeX is supported; plain \TeX support is possible, but it is more complicated and `ConTeXt` is not supported at all. In the following text we will focus only on \LaTeX .

`make4ht` supports a number of switches and options that affect the progress of compiling and processing of the output files. `make4ht` can be launched as follows:

```
make4ht <switches for make4ht> file.tex \
  "<options for tex4ht.sty"& \
  "<switches for tex4ht"& \
  "<switches for t4ht"& \
  "<switches for TeX"&
```

This complicated list is a result of the way `htlatex` works: it needs to pass options for all components involved in compilation. In most cases, fortunately, there is no need to use all the options. Most of the properties that `tex4ht` and `t4ht` provide can be requested using the `make4ht` switches.

3.1 `make4ht` command line switches

Every command line switch that `make4ht` supports has a short and long version. In addition, short switches can be combined. For example, the following two commands are identical:

```
make4ht --lua --utf8 --mode draft filename.tex
make4ht -lum draft filename.tex
```

This command uses \LuaTeX for the compilation, which will be executed only one once, due to `draft` mode. The resulting document will be in UTF-8 text encoding. The default output format for `make4ht` is HTML5 (`htlatex`'s default is HTML4).

In addition to these `--lua`, `--utf8`, and `--mode` switches, there are a number of other useful switches: `--config` (`-c`) configuration file for $\text{\TeX}4\text{ht}$, allowing tags inserted into output files to be changed.

```
--build-file (-e) select a build file.
--output-dir (-d) the directory where the output
  files will be copied.
--shell-escape (-s) pass the --shell-escape option
  to  $\text{\LaTeX}$ , enabling execution of external
  commands.
--xetex (-x) compile the document with  $\text{\XeTeX}$ .
--format (-f) select the output format.
```

There are other switches, but the above are the most commonly useful.

3.2 Output formats and extensions

$\text{\TeX}4\text{ht}$ supports a wide range of XML-based formats, from XHTML, through ODT to DocBook and TEI.

The `--format` switch for `make4ht` supports the formats `html5`, `xhtml`, `odt`, `tei` and `docbook` (the format names must be specified in lowercase). The default format is `html5`.

Formats can be selected also using the `tex4ht.sty` option:

```
make4ht filename.tex "docbook"
```

However, the advantage of `--format` is that it can fix some common issues for the particular formats. It can also load extensions. Extensions allow us to influence the compilation without having to use a build script. The list of extensions to be used can be written after the format name. They can be enabled using the plus character, and disabled with the minus character.¹ For example, the following command uses the `HTMLTidy` command to fix some common errors in the generated HTML file:

```
make4ht -f html5+tidy simple-example.tex
```

The following extensions are available:

`latexmk_build` Use the `latexmk` build system to compile the document. This will take care of calling external commands, for example, to create a bibliography.

`tidy` Clean HTML output with the `tidy` command.

`dvisvgm_hashes` efficient generation of images using the `dvisvgm` command. It can use multiple processors and only creates images that have been changed or created since the last compilation. This can make compilation noticeably faster.

`common_filters`, `common_domfilters` Clean the document using filters. Filters will be discussed later in the article.

`mathjaxnode` Convert MathML math code into special HTML using *MathJax Node Page*.² This

¹ Extensions can be enabled in a `make4ht` configuration file, so disabling them from the command line can be useful.

² github.com/pkra/mathjax-node-page

produces mathematics that can be viewed in web browsers without MathML support. The rendering of the result doesn't need JavaScript, which results in much faster display of the document compared to standard MathJax.

`staticsite` Create a document usable with static page generators such as *Jekyll*.³ These are useful for creating blogs or more complex websites.

3.3 Configuration files for `make4ht`

`make4ht` supports build scripts in the Lua language. They can be used to call external commands, to pass parameters to an executed command, to apply filters to the output files, to affect the image conversion, or to configure extensions.

The `.make4ht` configuration file is a special build script that is loaded automatically and should contain only general configurations shared between documents. In contrast, normal build files may contain configurations useful only for the current document. The configuration file can be located in the directory of the current document or in its parent directories.

This can be useful, for example, for maintaining a blog, with each document in its own directory. In the parent directory, a configuration file ensures proper processing. Here's a small example:

```
filter_settings "staticsite" {
  site_root = "output"
}
Make:enable_extension("common_domfilters")
if mode=="publish" then
  Make:enable_extension("staticsite")
  Make:htlatex {}
end
```

This configuration file sets the option `site_root` for the `staticsite` extension using the command `filter_settings`. This command can be used to set options for both filters and extensions. The name of the filter or extension is separated from the command by a space, followed by another space-separated field, where options can be set.

The next command is `Make:enable_extension`, which enables the extension. In this case the extension `common_domfilters` is used in every compilation, but the `staticsite` extension is used only in *publish* mode. In this mode it is also necessary to use the `Make:htlatex{}` to require at least one \LaTeX compilation.

Now we can run `make4ht` in publish mode:

```
make4ht -um publish simple-example.tex
```

The `output` directory will be created if it does not already exist; HTML and CSS files will be copied

³ jekyllrb.com

here. The static site generator must be configured to look for files here and it needs to be executed manually; the extension doesn't do that.

The resulting HTML looks something like this:

```
---
time: 1544811015
date: '2018-12-14 18:10:47'
title: 'sample'
styles:
- '2018-12-14-simple-example.css'
meta:
- charset: 'utf-8'
---
<p>Sample document</p>
```

The document header enclosed between the two `---` lines contains variables in the YAML format extracted from the HTML file. Only the contents of the document body remains in the document; the old header is stripped off. The static generator can then create a page based on the template and the variables in the YAML header.

This was just a basic example. Filters and extensions have much more extensive configurable options, all of which are described in the `make4ht` documentation.⁴

3.4 Build files

In the compilation scripts it is possible to use the same procedures as in the configuration file, but focused on the particular compiled document. The following code shows use of the DOM filters. These take advantage of the *LuaXML*⁵ library. It supports processing XML files using the Document Object Model (DOM) interface. This makes it easy to navigate, edit, create or delete elements.

The use of DOM filters is shown in the following example for \LaTeX :

```
\documentclass{article}
\begin{document}
Test {\itshape háčkú}
\end{document}
```

Because of a known error in processing the DVI file with the `tex4ht` command, each accented character in the generated HTML file will be placed in a separate `` element:

```
<!--1. 4--><p class="noindent" >Test
<span class="rm-lmri-10">h</span><span
class="rm-lmri-10">á</span><span
class="rm-lmri-10">č</span><span
class="rm-lmri-10">k</span></p>
```

The following build file removes this by using the built-in `joincharacters` DOM filter. In addition, it

⁴ ctan.org/pkg/make4ht

⁵ ctan.org/pkg/luaxml

changes the value of the `class` attribute for all `<p>` elements to `mypar`, just to show how to work with the DOM interface:

```
local domfilter = require("make4ht-domfilter")

local function domsample(dom)
  -- the following code will process
  -- all <p> elements
  for _, par in
    ipairs(dom:query_selector("p")) do
    -- set the "class" attribute
    par:set_attribute("class", "mypar")
  end
  return dom
end

local process = domfilter({
  "joincharacters",
  domsample
})
Make:match("html$", process)
```

The script uses the standard Lua `require` function to load the `make4ht-domfilter` library. This creates a `domfilter` function that takes a list of DOM filters to execute as a parameter.

Each call to the `domfilter` function creates another function with a chain of filters specified in a table. Parameters in the fields can be either the name of an existing DOM filter, or a function defined in the file.

The filter chain can then be used in the function `Make:match`. This takes a filename pattern to match files for which the filters should be executed, and the filter chain.

The `process` function will run on each file whose filename ends with `html` in this case.

The resulting HTML file does not contain the extra `` elements and the `<p>` element has a class attribute value of `mypar`:

```
<!-- 1. 3 --><p class='mypar'>
Test <span class='rm-lmri-10'>háčků</span>
</p>
```

Here is another example, of a more complex build file with external command execution and configuration of image generation:

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

The `Make:add` function defines a new usable command, `biber` in this case. The second parameter is a formatting string, which may contain `${...}` variable templates, which are in turn replaced by parameters set by `make4ht`. Here, the `${input}` will be replaced with the input file name.

The newly added command can then be used as `Make:<command>`, like the built-in commands. Additional variables may be set in the table passed as the argument.

The `Make:htlatex` command is built in and requires one execution of \LaTeX with `TeX4ht` active.

The `Make:image` command configures the image conversion. Three variables are available: `page` contains the page number of the image in the DVI file, `output` is the name of the output image, and `source` is the name of the `.idv` file.

The use of the `Make:match` command was shown in the previous example, but it may also contain a string with the command to be executed. The `filename` variable contains the name of the generated file currently being processed.

4 \TeX 4ht configuration

Output format tags embedded in a document are fully configurable via several mechanisms. The easiest way is to use the `tex4ht.sty` package options, a more advanced choice is to use a custom configuration file, and the most powerful option is to use `.4ht` files.

When a \TeX file is compiled using `make4ht` or another \TeX 4ht script, the `tex4ht.sty` package is loaded before the document itself. Package options are obtained from the compilation script arguments. As a result, it is not necessary to explicitly load the `tex4ht.sty` package in the document.

The \TeX file loading mechanism is modified to register each loaded file with `TeX4ht`. For some packages, `TeX4ht` has code to simply stop it from being loaded, or to immediately override some macros. This is necessary for packages that are incompatible with `TeX4ht`, such as `fontspec`.

After execution of the document preamble, the configuration files for the packages detected during processing are loaded. These files are named as the base filename of the configured package, extended with `.4ht`. Their main function is to insert configurable macros, called hooks, into the commands provided by the package. In general, it is better not to redefine macros, only to patch them with the commands `TeX4ht` provides for this purpose. This is enough in most cases.

Output format configuration files are loaded after the package configuration files are processed. These define the contents of the hooks. Besides

inserting output format tags, the hooks can contain any valid \TeX commands.

4.1 `tex4ht.sty` options

Many configurations are conditional, that is, executed only in the presence of a particular option being given for `tex4ht.sty`. Each output format configuration file can test any option, which means that there is no restriction on the list of possible options; each output format can support a different set of options.

As mentioned above, it is neither necessary nor desirable to load `tex4ht.sty` directly in the document, so it is possible to pass the options in other ways. The easiest way is to use the compilation script argument. This is always the argument following the document name. For example, here we specify the two options `mathml` and `mathjax`.

```
make4ht file.tex "mathml,mathjax"
```

Another way to pass options is to use `\Preamble` command in the private configuration file. We'll show this in the next section.

As mentioned above, the list of options is open-ended, but let's now look at some current options regarding mathematical outputs in HTML. The default configuration for mathematical environments produces a blend of rich text and images for more complex math, if it cannot be easily created with HTML elements. Often this output doesn't look good. As an alternative, it is possible to use images for all math content. This can be achieved by using the `pic-m` options for inline mathematics and `pic-(environment)` for mathematical environments. For example, the `pic-align` option will make images for all `align` environments.

By default, images are created in the PNG bitmap format. Higher quality can be achieved using the SVG vector format. This can be specified with the `svg` option.

The \TeX 4ht documentation is unfortunately somewhat spartan. With the `info` option, much useful information about the available configurations can be found in the `.log` file after the compilation of a document.

The options listed in the example above, `mathml` and `mathjax`, provide the best quality output for mathematical content. The MathML markup language, requested by the first option, encodes the mathematical information, but its support in Web browsers is poor. The second option requests the *MathJax* library, which can render the MathML output in all browsers with JavaScript support.

The `mathjax` option used without `mathml` completely turns off compilation of math by \TeX 4ht;

all math content remains in the HTML document as raw \LaTeX macros. MathJax then processes the document and renders the math in the correct way. The disadvantage of this method is that MathJax does not support all packages and user commands; it needs special configuration in these cases. Emulation of some complex macros may not even be possible.

4.2 Private configuration file

The private configuration file can be used to insert custom content into the configuration hooks. This file has a special structure:

```
<preamble definitions> ...
\Preamble{tex4ht.sty options}
... <normal configurations> ...
\begin{document}
... <configuration for HTML head>
\EndPreamble
```

The three commands shown here must be always included in this configuration file: `\Preamble`, `\begin{document}` and `\EndPreamble`. The configuration file name can be passed to `make4ht` using the switch `--config` (or `-c`), like this:

```
make4ht -c myconfig.cfg file.tex
```

The full path to the configuration file must be used if it is not placed in the current directory.

There are several configuration commands. The most important are `\Configure` for common configurations, `\ConfigureEnv` for configuration of \LaTeX environments, and `\ConfigureList` for the configuration of the list environments.

The `\HCode` command is used for insertion of the output format tags. The `\Hnewline` command inserts a newline in the output document. And the `\Css` command writes content to the CSS file.

The following example configures the hooks for the `\textit` command to insert the `` element.

```
\Configure{textit}
{\HCode{<em>}\NoFonts}
{\EndNoFonts\HCode{</em>}}
```

The `\Configure` command takes a variable number of arguments. It depends on the hooks' definition how many arguments are needed. The first argument is always the name of the configuration; following arguments then put the code in the hooks. Typically, a configuration requires two hooks: the first places code before the start of the command, the second after it is done. This is the case for the `textit` example above. The configuration name may match the name of the configured command, but this is not always the case. The package `.4ht` file may choose the configuration names arbitrarily.

The `\NoFonts` command used above disables inserting formatting elements for fonts when processing a DVI file. `TEX4ht` automatically creates basic formatting for font changes. This makes it possible to create a document with basic formatting even for unsupported commands, but it is not desirable when the command is configured using custom HTML elements.

Correct paragraph handling is difficult, and `TEX4ht` sometimes puts paragraph tags in undesired places. This applies primarily to environment configurations that can contain several paragraphs and yet enclose their entire content in one element. It may happen that the starting paragraph mark is placed before the beginning of this element, but it should be placed right after that. The `\IgnorePar` command can prevent the insertion of a tag for the next paragraph. `\EndP` inserts a closing tag for the previous paragraph. There are more commands to work with paragraphs, but these are the most important.

To illustrate this issue, the following example uses the hypothetical `rightaligned` environment:

```
\ConfigureEnv{rightaligned}
  {\HCode{<section class="right">}}
  {\HCode{</section>}}
  {}
  {}
```

The `\ConfigureEnv` command expects five parameters. The first is name of the environment to configure. The contents of the second parameter are inserted at the beginning of the environment, and the contents of the third at the end of the environment. The other two parameters are used only if the configured environment is based on a list. In most cases they may be left blank. The HTML code created by the configuration above will look something like the following:

```
<p class="indent" ><section class="right">
...
</p><p class="indent"></section>
```

As described above, this code is invalid. The terminating tag for the `<p>` element is placed at the wrong nesting level. The invalid code can cause the DOM filters and other post-processing tools expecting well-formed XML files to fail, so this situation must be avoided.

The correct configuration is somewhat more complicated:

```
\ConfigureEnv{rightaligned}
  {\ifvmode\IgnorePar\fi\EndP
   \HCode{<section class="right">}\par}
  {\ifvmode\IgnorePar\fi\EndP
   \HCode{</section>}}
```

```
{}
```

```
{}
```

In this case the insertion of tags for paragraphs is controlled, resulting in a correctly nested structure:

```
<section class="right">
<!--1. 9--><p class="indent" >
...
</p></section>
```

Another feature is the conversion of part of a document to an image. This can be requested using the commands `\Picture*` or `\Picture+`. The difference between these is that the first processes its content as a vertical box, and the second does not. The content between any of these commands and the closing `\EndPicture` is converted to an image.

The following example creates an image for the text contained in the `topicture` environment:

```
\documentclass{article}
\newenvironment{topicture}{\bfseries}{}
\begin{document}
\begin{topicture}
  Contents of this environment
  will be converted as an image.
\end{topicture}
\end{document}
```

The `TEX4ht` configuration for the `topicture` environment uses `\Picture*`:

```
\ConfigureEnv{topicture}
  {\Picture*{}}
  {\EndPicture}
  {}
  {}
```

The resulting document will contain an image of the text contained in the `topicture` environment as it was typeset in the DVI file.

5 Conclusion

The `TEX4ht` configuration options are extensive. We have touched only the basics in this article, but it should be enough to solve many basic issues that users might face. We omitted examples of how to add configurations for a new `LATEX` package; we hope to address this topic in a future article.

The system is easier and more efficient to use than in the past, thanks to the `make4ht` build system.

The new documentation for `TEX4ht` is being developed with financial support by `CSTUG`. It will describe the most useful user configurations, as well as technical details of the system.

- ◇ Michal Hoftich
Charles University, Faculty of Education
michal.hoftich (at) pedf dot cuni dot cz
<https://www.kodymirus.cz>

TUGboat online, reimplemented

Karl Berry

Abstract

This article discusses updates to the data and code for creating the online *TUGboat* HTML files which are automatically generated: the per-issue tables of contents and the accumulated lists across all issues of authors, categories, and titles. All source files, both data and code, are available from <https://tug.org/TUGboat>, and are released to the public domain.

1 Introduction

Since 2005, *TUGboat* has had web pages generated for both per-issue tables of contents and accumulated lists across all issues of authors, categories, and titles. David Walden and I worked on the process together and wrote a detailed article about it [1]; Dave wrote all of the code. More recently, we wanted to add some features which necessitated writing a new implementation. This short note describes the new work.

The basic process remains unchanged. To briefly review from the earlier article:

- For each issue, a source file `tb<n>capsule.txt` (n being the *TUGboat* issue sequence number), which is essentially written in \TeX (it is used to create the contents by difficulty on the inside back cover), is converted to an HTML file named `contents<vv-i>.html` (for issue number i in volume vv). These `contents*.html` files are intended to closely mimic the printed table of contents (the back cover) with respect to ordering of items, variation in author’s names, category names, etc., with only typos corrected.
- The translation from \TeX to HTML is done by the code here, not using `\TeX4ht` or any other tool; the overall HTML structure is written directly by the program. The translation is informed by two files (`lists-translations.txt` and `lists-regexps.txt`), which (simplistically) map \TeX input strings to HTML output strings.
- Finally, three files are produced accumulating all items from across all issues: `listauthor.html`, `listkeyword.html`, and `listtitle.html`; each is grouped and sorted accordingly. (These cumulative lists are the primary purpose for developing the program in the first place.) In these files, in contrast to the per-issue contents, many unifications are done (directed by a third external data file, `lists-unifications.txt`), so that articles written under the names, say, “Donald E. Knuth”, “Donald Knuth”, “Don Knuth”, etc., all appear together. Similarly, many varia-

tions in category names, and related categories, are merged.

- The translations are applied first, then the regular expressions (regexps), and finally the unifications.

2 General implementation approach

Both the old implementation and the new are written in Perl, though they do not share any code. I chose Perl simply because it is the scripting language in which I am most comfortable writing nowadays. There was no need to use a compiled language; the total amount of data is small by modern standards. Readability and maintainability of the code are far more important than efficiency.

I wrote the new implementation as a straightforward, if perhaps old-fashioned, program. I did not see the need to create Perl modules, for example, since the program’s job is a given, and the chance of any significant reuse outside the context of *TUGboat* seems small indeed. All the code and data are released to the public domain, so any subroutines, utility functions, fragments, or any other pieces of code or data useful elsewhere can be copied, modified, and redistributed at will.

As mentioned above, the capsule source files are essentially \TeX . For example, here is the capsule entry from `tb123capsule.txt` for a recent article:

```
\capsule{
  {Electronic Documents}%add|Software \& Tools
  {Martin Ruckert}
  {\acro{HINT}: Reflowing \TeX\ output}
  {postponing \TeX\ page rendering to ...}
  {217-223}
  {/TUGboat/!TBIDENT!ruckert-hint.pdf}
```

The meaning of the fields is described in the previous article, but is probably evident enough just from the example. For our present purposes, let’s just observe the brace-delimited arguments and general \TeX markup. To parse this, the present program uses one non-core Perl module (and only this one): `Text::Balanced` (metacpan.org/pod/Text::Balanced), which does basic balanced-delimiter parsing. (The previous implementation did the parsing natively, more or less line-based.)

Perl has several modules to do this job; I chose this one because (a) it had a reasonably simple interface, and (b) it could return the non-balanced text between arguments, which was crucial for our format, since we use formatted comments as directives with additional information for the `lists*` files — as seen above with the `%add|...` extra category. Only three directives have been needed so far: to add and replace categories, and to add authors. They are

crucial for making the accumulated lists include all the useful information.

Each capsule turns into a Perl hash (associative array), and each issue is another hash, including pointers to all its capsules, and so on. In general, the amount of data is so small that memory usage was a non-issue.

Perhaps it would be a better general approach to completely reformat the \TeX source into a non- \TeX format (YAML, for instance) and then parse that; and perhaps some future *TUGboat* worker will feel inspired to do that. It would not be especially hard to have the current implementation output such a conversion as a starting point. I merely chose to keep the process more or less as it has been.

3 Cleaning up capsule sources and output

I did take the opportunity to clean up the capsule source files, e.g., using more consistent macro abbreviations, adding missing accents to authors' names, correcting typos, etc. The balanced-brace parsing regime meant that unbalanced braces got found, of which there were several.

I also added consistency checks in the code, so that, for example, a new category name that we happen to invent for a future issue will get reported; such cases should (probably) be unified with an existing category. Many unifications of existing categories and authors were also added.

Another part of the cleanup was to ensure that the page number of each item is unique; when two items start on the same page, internally we use decimals (100 and 100.5, say) to order them. This is done with a macro `\offset`. Naturally such decimals are not shown in either the \TeX or HTML output. They are necessary in order to have a unique key in all our various hashes, and for sorting.

Speaking of sorting, I wanted the new output for the accumulated lists to be stably sorted, so that the results of any code or data changes could be easily diffed against the previous output. So now the sorting for a given entry is reliably by volume, then issue, then page (and first by title for the title list); having unique internal page values was also a prerequisite for the stable sort.

Another minor point about the HTML output is the anchor names: we intentionally reduce all anchor identifiers (author names, titles, etc.) to 7-bit ASCII — indeed, only letters, numbers, periods, and commas. For example, Herbert Voß's items in *TUGboat* are available at `tug.org/TUGboat/Contents/listauthor.html#Voss,Herbert`. (Sorry, Herbert.) Similarly, if an anchor starts with a non-letter, it is prefixed by `t_`. Although HTML permits general Uni-

code in anchor names nowadays, this has not always been the case, and regardless, for ease of copying, use in email, etc., this seemed the most useful approach.

4 Data files `lists-*.txt`

The external data files `lists-unifications.txt` and `lists-translations.txt` that play a part in all these conversions are described in the earlier article. The third file mentioned above, `lists-regexps.txt`, is new in this implementation. Here are some example entries from each.

4.1 `lists-unifications.txt`

Examples from `lists-unifications.txt`:

```
Dreamboat
  Expanding Horizons
  Future Issues
...
Max D&iacute;az
  M. D&iacute;az
```

The left-justified line shows the name as it should be shown, and following indented lines show alternate names as they are found. We unify categories and names, as shown here.

The Díaz example also shows that we do unifications after the translation to HTML, so both the canonical name and the alternates are expressed that way, not in \TeX . Thus, the exact form of the translation matters (whether `í` translates to `í` or `í` or `í` or a literal UTF-8 `í` or ...) and has to match with the `lists-translations.txt` entries. Examples from there are next.

4.2 `lists-translations.txt`

Examples from `lists-translations.txt`:

```
\'{i}||&iacute;||i
\TUG{}||TUG
```

Each line is two or three strings, separated by a `||` delimiter. The first element is what's in the \TeX source; the second is the HTML to output, and the third is the plain text conversion for sorting and anchors. If the third element is absent (as in the `\TUG` line above), the second element is used for both plain text and HTML.


4.3 `lists-regexps.txt`


For `lists-regexps.txt`, the general form is similar to `lists-translations.txt`, with a left-hand side and right-hand side separated by the same `||` delimiter. But here, the lhs is a regular expression, and the rhs is a replacement expression:

```
\\emph\{(.*)\}||"<i>$1</i>"
\{\it\s*(.*)\}||"<i>$1</i>"
```

All that punctuation may look daunting, but if taken a bit at a time, it is mostly standard regular expression syntax. The above two entries handle the usual L^AT_EX `\emph{...}` and plain `{\it ...}` italic font switching (with no attempt to handle nested `\emph`, as it is not needed). Both syntaxes for font switching are prevalent throughout the capsule sources.

The `.*?` construct in the left hand side may be unfamiliar; this is merely a convenience meaning a “non-greedy” match — any characters up until the first following right brace (the `\}` means a literal right brace character). A `.*` without the `?` would match until the *last* following right brace.

 On second glance, what also may seem unusual is the rhs being enclosed in double quotes, `"..."`, specifying a Perl string constant. Why? Because, ultimately, this is going to turn into a Perl substitution, `s/<lhs>/<rhs>/g` (all substitutions specified here are done globally), but initially the lhs and rhs have to be read into variables — in other words, string values. But we don’t want to evaluate these strings when they are read from the `lists-regexps` file; the `$1` in the rhs needs to refer to what is matched by the `(...)` group on the lhs when the substitution is executed. This turns out to be a programming exercise in layers of evaluation.

 The simplest way I found to do it was to use a Perl feature I had never before needed, or even heard of, in my 30-odd years of using Perl since it first appeared: including the `/ee` modifier on the substitution, as well as the `/g`. I won’t try to explain it here; for the curious, there is a discussion at stackoverflow.com/q/392644, in addition to the Perl manual itself (`perlre`).

5 Performance and profiling

Although I said above that efficiency was not an issue, that is not quite true. Especially during development and debugging, doing a test run must not take too long, since it gets done over and over. My extremely naive initial version took over 45 seconds (on my development machine, which is plenty fast) to process the ≈ 120 TUGboat issues — much too frustrating to be borne.

The Perl module `Devel::NYTProf` turned out to be by far the best profiling tool available. (By the way, NYT stands for *New York Times*; a programmer there did the initial development.) Unlike other Perl profiling tools, it shows timing data per individual source line, not just functions or blocks.

Using that, it turned out that almost all the time was being consumed dealing with the substitutions from the `lists-*` files, since the strings were being read at runtime, instead of being literal in

the source code. The easy step of “precompiling” the regular expressions after reading the files, with `qr//`, resulted in the total runtime dropping an order of magnitude, to under 4 seconds. So development could proceed without any major surgery on the code or data structures.

6 Conclusion

The ad hoc conversion approach described here is viable only because we have complete control over the not-very-complicated input, and desirable mainly because we want complete control over the output. I did not want to struggle with any tool to get the HTML I wanted, namely to be reasonably formatted and otherwise comprehensible, and not making significant use of external resources or JavaScript.

Although changes and new needs are inevitable, I hope the program and data will be sufficiently robust for years to come.

References

- [1] K. Berry and D. Walden. TUGboat online. *TUGboat* 32(1):23–26, 2011. <http://www.tug.org/TUGboat/tb32-1/tb100berry.pdf>

◇ Karl Berry
<https://tug.org/TUGboat>

Editor’s note: Until last year, the comprehensive B^IB^TE^X database for TUGboat, `tugboat.bib`, which Nelson Beebe maintains on the servers at the University of Utah was derived programmatically from the TUGboat contents files. Earlier this year, Nelson modified the procedures to instead use the HTML files (generated as described here) for the issues. This method should not only be easier to maintain, but also contain additional information.

New labeling conventions have been applied to newly created entries; these follow Nelson’s BibNet Project. Labels of older entries are frozen in the old form (e.g., `Knuth:TB2-3-5`), so as not to invalidate user documents that cite such labels.

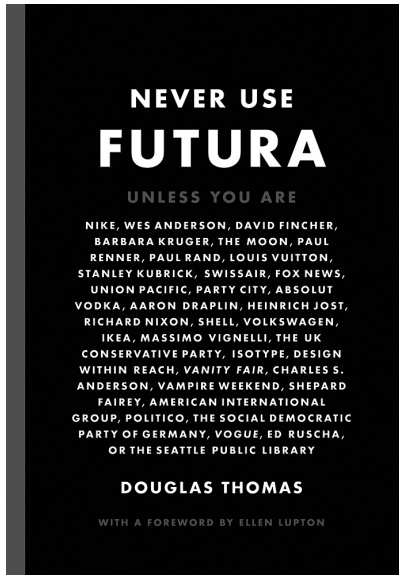
These files are on both the Utah web server, at www.math.utah.edu/pub/tex/bib, and the parallel FTP server. In addition to `tugboat.bib`, these associated files are present: `.def`, `.dvi`, `.html`, `.ltx`, `.pdf`, `.ps.gz`, `.ps.xz`, `.sok`, and `.twx`. (To fetch the whole collection under `ftp`, issue the command “`mget tugboat.*`”.)

The collection is also on CTAN in the area `info/biblio`, along with `bib` files on additional topics. Some of these, including `tugboat.bib`, are also in T_EX Live. — Barbara Beeton

**Book review: *Never use Futura*,
by Douglas Thomas**

Boris Veysman

Douglas Thomas, *Never use Futura*. Princeton Architectural Press, 2017, paperback, 208pp., US\$24.95, ISBN 978-1616895723.



The only extraterrestrial body visited by humans has a plaque with the proud words WE CAME IN PEACE FOR ALL MANKIND and signatures of Neil Armstrong, Michael Collins, Edwin Aldrin, & Richard Nixon. This plaque is typeset in Futura (Figure 1), a typeface created by German designer Paul Renner in 1927 and popularized by a German company Bauer Type Foundry. Four decades later the typeface was so ubiquitous, that its use on NASA documents, including the famous plaque, became a matter of course: astronauts were required to quickly read and understand instructions, so a familiar readable typeface was chosen. Today this ubiquitousness works *against* the typeface: it is often considered overused, prompting the advice *Never use Futura* by some experts. This advice is boldly used as a title by a book by Douglas Thomas — followed by the sly *unless you are*, and a long list of distinguished companies on the title page.

The book is based on the author’s Master’s thesis at the University of Chicago and is a well-researched and generously illustrated treatise on the history of Futura. One of the most important questions Thomas tries to answer in his book is why the typeface became so popular. He argues that Futura fortuitously combined the bold modernist simplicity of geometric sans serif and a number of well-thought-



Figure 1: The moon plaque

out features based on the traditions of font design. Among these features are the classical proportions of the typeface; the subtle variations of height (the sharp tops of capital A and W are slightly above the other capital letters to create a visual uniformity of height) and pen width (the bowls of a, d, p become slightly thinner when they touch the stems). These almost imperceptible features make the typeface alive rather than mechanistic and cold.

The author is fascinated by the way the history of Futura is intertwined with the history of the 20th century. Due to its German origins, Futura was an object of propaganda in the USA: “By buying German fonts you help Nazis”, as American foundries that sold their own clones of Futura wrote in their advertisements. Nevertheless, as Thomas shows, many US Army materials were set in Futura and other typefaces of German origin. In Germany the fate of Futura was more complicated. The Nazi regime disliked Bauhaus and modernist art. The anti-fascist views of Paul Renner did not help either — the font designer was briefly arrested after Hitler took power (later he was able to emigrate to Switzerland).

Up to 1941 the official Nazi position on typography was that only black letter fonts were truly Aryan. However, in 1941 there was an abrupt change in the policy: black letter fonts were declared a Jewish corruption, and Germans were told to use only Roman fonts. This story is discussed in a paper by Yannis Haralambous (*Typesetting Old German: Fraktur, Schwabacher, Gotisch and Initials*, *TUGboat*, 12:1, 129–138, 1991, tug.org/TUGboat/tb12-1/tb31hara.pdf). By the way, this paper reproduces NSDAP order 2/41 introducing the change in the font policy. Interestingly enough, the letterhead of the order is typeset in black letter. At any rate, many later Nazi documents are printed in Futura. Thomas shows the identification card of a

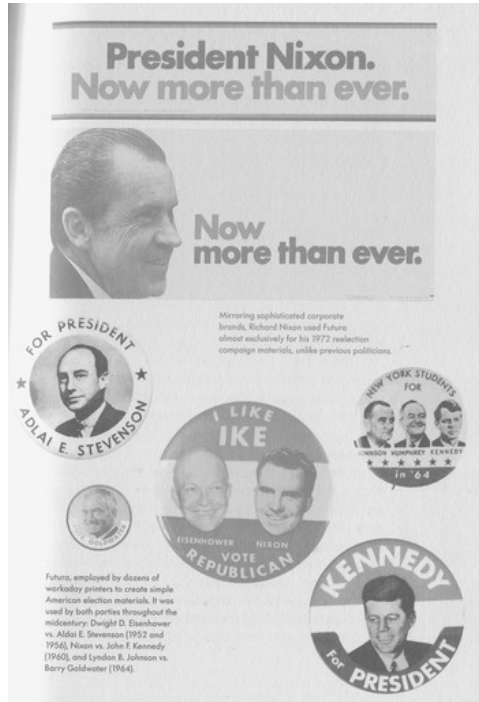


Figure 2: Futura in electoral politics

ghetto Jew typeset in Futura. Poignantly, several pages later he reproduces the proclamation about the internment of Japanese Americans typeset in the same font.

The general feel of Futura, its combination of modernity and familiarity, made it ideal for electoral copy. At one point almost all electoral materials were typeset in Futura or its clones; see Figure 2. In our more sophisticated times, top ticket candidates usually order bespoke typefaces for their campaigns—often a version of geometric sans serif. In the 2016 US presidential elections, Hillary Clinton had a custom made typeface *Unity* based on Sharp Sans, while many Republican candidates used Futura-based fonts (Chris Christie, Marco Rubio, Jeb Bush). Interestingly enough, the campaign of candidate Trump did not make any design choice, so Trump posters in different places used quite different fonts, mostly bold-face. While the word TRUMP was most often typeset in Akzidenz-Grotesk Bold Extended, sometimes Microsoft’s default Arial was used. The typeface for the phrase “Make America Great Again” on Trump hats is the blandest Times New Roman, “almost an antichoice”, writes Thomas.

While the book has many similar sociological anecdotes, it also has many things one expects from a book about fonts, for example, a comparison of different digital versions of Futura (see, for example, the tracing of the letter S in Figure 3). It thor-

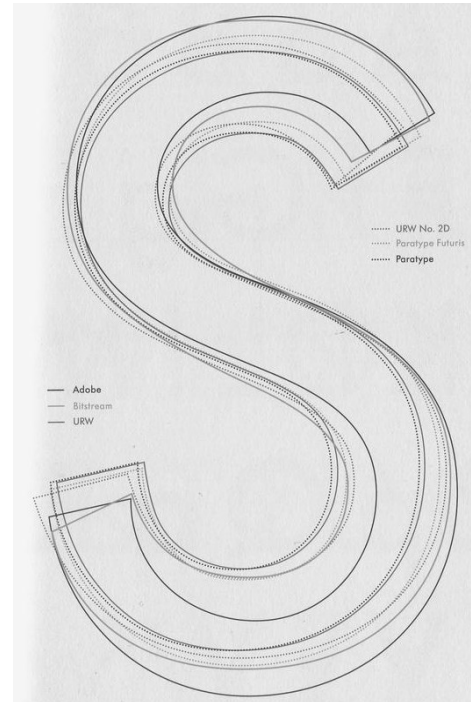


Figure 3: ‘S’ in several digital versions of Futura

oughly explores Futura’s many uses, including art and advertising (Figure 4).

Included in the book is a fascinating photo essay *Futura in the wild*. An example of Futura in the mall is accompanied with an interesting observation that lighter faces subtly indicate more expensive wares (see Figure 5).

The book has a useful index and expansive end notes with the bibliography, showing its provenance in academic research. It is well designed and competently typeset. The body text is set in a serif font Lyon from Commercial Type. The chapter headings are typeset in Futura ND Bold from Neufville Digital, while captions are in Futura SB-Medium from Scangraphic. The foreword is written by Ellen Lupton, the author of a number of interesting books including *Thinking with type*. Even the blurbs on the back cover are creative: they are “authored” by various typefaces themselves.

This book is a good addition to any typography lover’s library.

Acknowledgment. This review was suggested by Dave Walden, who sent me his copy of the book.

◇ Boris Veytsman
Systems Biology School, George
Mason University, Fairfax, VA
borisv (at) lk dot net
<http://borisv.lk.net>

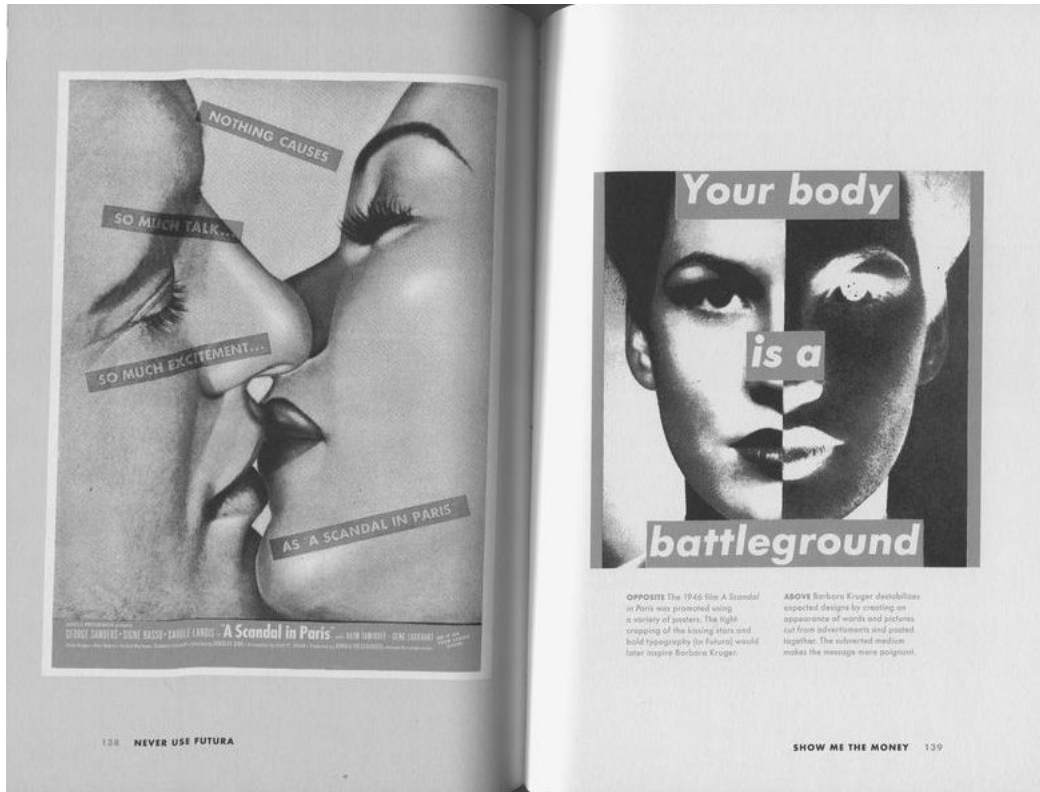


Figure 4: Posters with Futura



Figure 5: In the mall, from the photo essay *Futura in the wild*

Die \TeX nische Komödie 1/2019

Die \TeX nische Komödie is the journal of DANTE e.V., the German-language \TeX user group (dante.de). (Non-technical items are omitted.)

CHRISTOPH GRÖNINGER, \LaTeX -Compiler in CMake-Projekten verwenden [Using a \LaTeX compiler in CMake-Projects]; pp. 13–17

In larger C++ projects one often uses the CMake build system to invoke the compiler and linker in different settings. If there is \LaTeX -based documentation it will probably be compiled with CMake as well. For this purpose the two projects `UseLATEX.cmake` and `UseLatexMk` were created. This article describes their use.

TACO HOEKWATER, Wie installiere ich eine Schrift für Con \TeX t? [How to install a font for Con \TeX t?]; pp. 17–30

Installing a new font family for Con \TeX t is not especially difficult for experienced users, but newbies usually struggle a little. This article describes the process based on the freely available IBM Plex font.

HERBERT VOSS, Rekursive Aufrufe am Beispiel von Abkürzungen [Using recursion for abbreviations]; pp. 31–34

Recursion is a commonly used strategy to solve problems in programming. Since \TeX is Turing-complete, recursion can be used here as well. In this article we show the process with an example to generate abbreviations.

MARKUS KOHM, Ergänzung zum Beitrag „KOMA-Script für Paketautoren . . .“ [Supplement to the article “KOMA-Script for package authors . . .”]; pp. 34–36

This article is a supplement to the mentioned article in DTK 4/2018. Due to some adjustments in the `nomenc1` package there are some changes described in this article.

HERBERT VOSS, Schriften für mehrsprachige Texte [Fonts for multi-lingual texts]; pp. 37–38

The Noto font, developed by Google, is available in so many versions that the compressed archive has more than one gigabyte. In a \TeX Live or Mi \TeX full installation, the most important fonts are included as OpenType fonts.

HERBERT VOSS, Garamond-Math; pp. 38–40

With Garamond-Math there is a new OpenType font for typesetting mathematical content.

[Received from Herbert Voß.]

Zpravodaj 2018/1–4

Zpravodaj is the journal of $\mathcal{C}\mathcal{S}\mathcal{T}\mathcal{U}\mathcal{G}$, the \TeX user group oriented mainly but not entirely to the Czech and Slovak languages (cstug.cz).

VÍT NOVOTNÝ, Příprava Zpravodaje $\mathcal{C}\mathcal{S}\mathcal{T}\mathcal{U}\mathcal{G}$ [Preparing the $\mathcal{C}\mathcal{S}\mathcal{T}\mathcal{U}\mathcal{G}$ Bulletin]; pp. 1–10

The article describes the structure, the typesetting and the preflight of the $\mathcal{C}\mathcal{S}\mathcal{T}\mathcal{U}\mathcal{G}$ Bulletin. We take a detailed look at the journey of a manuscript to the readers' mailboxes. The author has been the editor of the $\mathcal{C}\mathcal{S}\mathcal{T}\mathcal{U}\mathcal{G}$ Bulletin since 2016.

MICHAEL HOFNICH, Publikování z \LaTeX u na web pomocí \TeX 4ht [LaTeX to web publishing using \TeX 4ht]; pp. 11–21
[Reprinted in this issue of *TUGboat*.]

MAREK POMP, Tabulky v dobře dokumentovaných statistických výpočtech [Tables in well-documented statistical calculations]; pp. 22–37

The article describes the method of publishing well-documented statistical calculations using the R software. It is especially about creating tables using `knitr` and `kableExtra`.

HANS HAGEN, Lua \TeX version 1.0.0; pp. 38–42
[Printed in *TUGboat* 37:3.]

HANS HAGEN, Emoji again; pp. 43–58

Since the 10th International Con \TeX t Meeting in 2016, Con \TeX t has supported the OpenType `colr` and `cpal` tables that are used in color fonts and also to produce emoji. The article introduces emoji and uses the Microsoft's `seguiemj` font to show how emoji are constructed from glyphs, how emoji can be stacked into sequences, and how the palettes of a color font can be changed in Con \TeX t.

HANS HAGEN, Con \TeX t performance; pp. 59–78

The processing speed of a \TeX engine is affected by a number of factors, such as the format, macros, scripting, fonts, microtypographic extensions, Sync \TeX , and command-line redirection. The article discusses the individual factors from the perspective of a Con \TeX t user. The article also measures the overhead of Con \TeX t MkII and MkIV, the impact of command-line redirection and fonts on the speed of Con \TeX t MkIV, and the overall speed of typesetting with Con \TeX t MkII and MkIV.

HANS HAGEN, Variable fonts; pp. 79–89
[Printed in *TUGboat* 38:2.]

PETER WILSON, Glisterings #8: It Might Work. VII – Macros; pp. 90–100
[Printed in *TUGboat* 29:2.]

Translated to Czech by Jan Šustek.]

[Received from Vít Novotný.]



The Treasure Chest

This is a selection of the new packages posted to CTAN (ctan.org) from October 2018–April 2019, with descriptions based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. More information about any package can be found at ctan.org/pkg/pkgname. A few entries which the editors subjectively believe to be of especially wide interest or otherwise notable are starred (*); of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the T_EX community. See also ctan.org/topic. Comments are welcome, as always.

◇ Karl Berry
tugboat (at) tug dot org

biblio

`alpha-persian` in `biblio/bibtex/contrib`
Persian version of `alpha.bst`, using X_YT_EX.
`econ-bst` in `biblio/bibtex/contrib`
Customizable BIB_TE_X style for economics papers.
`zootaxa-bst` in `biblio/bibtex/contrib`
BIB_TE_X style for the journal *Zootaxa*.

fonts

`crimsonpro` in `fonts`
Eight weights and italics for each weight.
`cuprum` in `fonts`
Cuprum font family support for L^AT_EX.
`garamond-math` in `fonts`
OpenType math font matching EB Garamond.
`inriafonts` in `fonts`
Inria fonts with L^AT_EX support.

graphics

`chordbars` in `graphics/pgf/contrib`
Chord grids for pop/jazz tunes in TikZ.
`chordbox` in `graphics/pgf/contrib`
Draw chord diagrams in TikZ.
`euflag` in `graphics`
Flag of European Union using standard packages.
`fiziko` in `graphics/metapost/contrib/macros`
MetaPost library for physics textbook illustrations.
`memorygraphs` in `graphics/pgf/contrib`
Typeset graphs of program memory in TikZ.
`ptolemaicastronomy` in `graphics/pgf/contrib`
Diagrams of sphere models for variably strict conditionals (Lewis counterfactuals).

`pst-moire` in `graphics/pstricks/contrib`
Moiré patterns in PSTricks.
`pst-venn` in `graphics/pstricks/contrib`
Venn diagrams in PSTricks.
`tikz-imagelabels` in `graphics/pgf/contrib`
Put labels on existing images using TikZ.
`tikz-truchet` in `graphics/pgf/contrib`
Draw Truchet tiles in TikZ.
`tikzlings` in `graphics/pgf/contrib`
Cute little animals and similar creatures in TikZ.

indexing

* `xindex` in `indexing`
Unicode-compatible index generation (in Lua).

info

* `joy-of-tex` in `info`
Text of Michael Spivak's *Joy of T_EX*, for the AMS-T_EX format.
`latex4musicians` in `info`
Guide for combining L^AT_EX and music.
`texonly` in `info`
Sample document in plain T_EX.

language/thai

`zhlineskip` in `language/chinese`
Line spacing for CJK documents.

language/japanese

`bxjaholiday` in `language/japanese`
Support for Japanese holidays.
`pxjodel` in `language/japanese`
Help change metrics of fonts from `japanese-otf`.

macros/generic

`poormanlog` in `macros/generic`
Standalone package for logarithms and powers, with almost 9 digits of precision.

macros/latex/contrib

`armymemo` in `macros/latex/contrib`
Class for Army memorandums, following AR 25-50.
`asmeconf` in `macros/latex/contrib`
Template for ASME conference papers.
`brandeis-problemset` in `macros/latex/contrib`
Class for COSI problem sets at Brandeis University.
`bussproofs-extra` in `macros/latex/contrib`
Extra commands for `bussproofs.sty`.
`changelog` in `macros/latex/contrib`
Changelog environment; supports multiple authors, unreleased changes, revoked releases, etc.
`commedit` in `macros/latex/contrib`
Commented (teacher/student) editions.

`macros/latex/contrib/commedit`

- `elegantbook` in `macros/latex/contrib`
 “Elegant” (Chinese) template for books.
- `elegantnote` in `macros/latex/contrib`
 “Elegant” (Chinese) template for notes.
- `elegantpaper` in `macros/latex/contrib`
 “Elegant” (Chinese) template for economics papers.
- `els-cas-templates` in `macros/latex/contrib`
 Typeset articles for Elsevier’s Complex Article Service (CAS) workflow.
- `eqexpl` in `macros/latex/contrib`
 Align explanations for formulas.
- `exam-randomizechoices` in `macros/latex/contrib`
 Random order of choices, under the `exam` class.
- `exercisepoints` in `macros/latex/contrib`
 Count and score exercises.
- `exframe` in `macros/latex/contrib`
 Framework for exercise problems.
- `fascicules` in `macros/latex/contrib`
 Create mathematical manuals for schools.
- `fbox` in `macros/latex/contrib`
 Extended `\fbox` macro from standard L^AT_EX.
- `frenchmath` in `macros/latex/contrib`
 Typesetting mathematics with French rules.
- `ftc-notebook` in `macros/latex/contrib`
 Typeset FIRST Tech Challenge notebooks.
- `gammas` in `macros/latex/contrib`
 Template for the GAMM Archive for Students.
- `gitver` in `macros/latex/contrib`
 Typeset current git hash of a document.
- `globalvals` in `macros/latex/contrib`
 Declare global variables that can be used anywhere, including before their declaration.
- `glossaries-estonian` in `macros/latex/contrib`
 Estonian translations for the `glossaries` package.
- `icon-appr` in `macros/latex/contrib`
 Create icon appearances for form buttons.
- `invoice-class` in `macros/latex/contrib`
 Make a standard US invoice from a CSV file.
- `iodhbwm` in `macros/latex/contrib`
 Unofficial template of the DHBW Mannheim.
- `identkey` in `macros/latex/contrib`
 Bracketed dichotomous identification keys.
- `keyindex` in `macros/latex/contrib`
 Index entry by key lookup, e.g., for names.
- `latex-uni8` in `macros/latex/contrib`
 Generic `inputenc`, `fontenc`, and `babel` for pdfL^AT_EX and LuaL^AT_EX.
- `latexalpha2` in `macros/latex/contrib`
 Embed Mathematica code and plots into L^AT_EX.
- `latexcolors` in `macros/latex/contrib`
 Color definitions from `latexcolors.com`.
- `lectures` in `macros/latex/contrib`
 Support for typesetting lecture notes.
- `lstfiracode` in `macros/latex/contrib`
 Use the Fira Code font for listings.
- `ltxguidex` in `macros/latex/contrib`
 Extended `ltxguide` class.
- `*metalogox` in `macros/latex/contrib`
 Adjust T_EX logos, with font detection.
- `mi-solns` in `macros/latex/contrib`
 Extract solutions from exercises and quizzes.
- `mismath` in `macros/latex/contrib`
 Mathematical macros for ISO rules.
- `modeles-factures-belges-assocs` in `m/l/c`
 Make invoices for Belgian non-profit organizations.
- `multicolrule` in `macros/latex/contrib`
 Decorative rules between columns.
- `njurepo` in `macros/latex/contrib`
 Report template for Nanjing University.
- `qsharp` in `macros/latex/contrib`
 Syntax highlighting for Q# language.
- `realhats` in `macros/latex/contrib`
 Replace math hat (\hat{u}) symbols with pictures of actual hats.
- `rgltxdoc` in `macros/latex/contrib`
 Common macros for documentation of the author’s packages.
- `rulerbox` in `macros/latex/contrib`
 Draw rulers around a box.
- `ryersongstheiss` in `macros/latex/contrib`
 Thesis template for the Ryerson School of Graduate Studies (SGS).
- `scratch3` in `macros/latex/contrib`
 Draw Scratch programs with L^AT_EX.
- `tblvar` in `macros/latex/contrib`
 Tables of signs and variations per French usage.
- `topiclongtable` in `macros/latex/contrib`
 Extend `longtable` with cells that merge hierarchically.
- `ucalgmthesis` in `macros/latex/contrib`
 Thesis class for University of Calgary Faculty of Graduate Studies.
- `xcpdftips` in `macros/latex/contrib`
 Extend `natbib` citations with PDF tooltips.
-
- macros/latex/contrib/beamer-contrib/themes**
- `beamerauxtheme` in `m/l/c/b-c/themes`
 Supplementary outer and inner themes for beamer.
- `beamertheme-light` in `m/l/c/b-c/themes`
 Minimal beamer theme.
-
- macros/latex/contrib/biblatex-contrib**
- `icite` in `m/l/c/biblatex-contrib`
 Indices locorum citatorum: indexes of authors and works, generated from a bibliography.
- `windycity` in `m/l/c/biblatex-contrib`
 A Chicago style for BIBL^AT_EX.
-
- macros/luatex/latex**
- `beamer-rl` in `macros/luatex/latex`
 Right-to-left presentation with `beamer` and `babel`.
- `luaimageembed` in `macros/luatex/latex`
 Embed images as base64-encoded strings.
- `luarandom` in `macros/luatex/latex`
 Create lists of random numbers.

makecookbook in macros/luatex/latex
Support cookbook typesetting.

macros/xetex/latex

quran-de in macros/xetex/latex
German translations for the quran package.
technion-thesis-template in macros/xetex/latex
Thesis template for the Technion graduate school.
tetragonos in macros/xetex/latex
Macro mapping for Chinese characters for the four-corner method.

macros/xetex/plain

do-it-yourself-tex in macros/xetex/plain
Modifiable forms, macros, samples for plain X_ƎTeX.

support

pdftex-quiet in support
Filter and colorize pdftex terminal output.
pkgcheck in support
CTAN package checker.

systems/unix

tex-fpc in systems/unix
Change files for the Free Pascal compiler.

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 tug.org/consultants.html. If you'd like to be listed, please see there.

Aicart Martinez, Mercè

Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: [m.aicart \(at\) ono.com](mailto:m.aicart@ono.com)
Web: <http://www.edilatex.com>

We provide, at reasonable low cost, L^AT_EX or T_EX 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

+1 213-617-8483
Email: [typesetting \(at\) dangerouscurve.org](mailto:typesetting@dangerouscurve.org)

We are your macro specialists for T_EX or L^AT_EX fine typography specs beyond those of the average L^AT_EX macro package. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical T_EX and L^AT_EX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

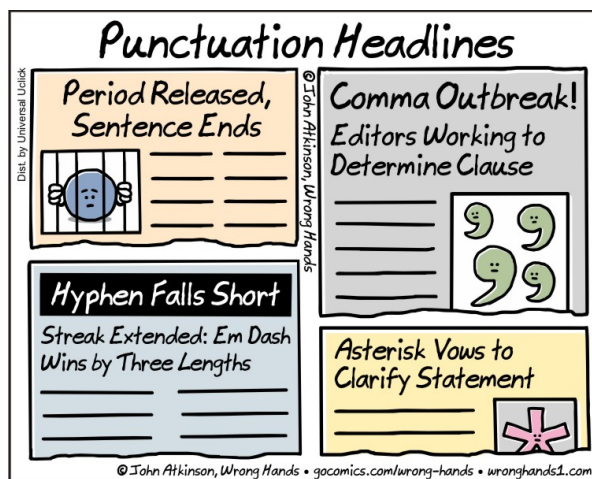
Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a T_EX book.

Dominici, Massimiliano

Email: [info \(at\) typotexnica.it](mailto:info@typotexnica.it)
Web: <http://www.typotexnica.it>

Our skills: layout of books, journals, articles; creation of L^AT_EX classes and packages; graphic design; conversion between different formats of documents.

We offer our services (related to publishing in Mathematics, Physics and Humanities) for documents in Italian, English, or French. Let us know the work plan and details; we will find a customized solution. Please check our website and/or send us email for further details.



Comic by John Atkinson (<https://wronghands1.com>).

Latchman, David

2005 Eye St. Suite #6
 Bakersfield, CA 93301
 +1 518-951-8786
 Email: david.latchman (at) texnical-designs.com
 Web: <http://www.texnical-designs.com>

L^AT_EX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized L^AT_EX packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

Sofka, Michael

8 Providence St.
 Albany, NY 12203
 +1 518 331-3457
 Email: michael.sofka (at) gmail.com

Personalized, professional T_EX and L^AT_EX consulting and programming services.

I offer 30 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in T_EX and L^AT_EX: Automated document conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in T_EX or L^AT_EX, `knitr`.

If you have a specialized T_EX or L^AT_EX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

T_EXtnik

Spain

Email: [textnik.typesetting \(at\) gmail.com](mailto:textnik.typesetting@gmail.com)

Do you need personalised L^AT_EX class or package creation? Maybe help to finalise your current typesetting project? Any problems compiling your current files or converting from other formats to L^AT_EX? We offer +15 years of experience as advanced L^AT_EX user and programmer. Our experience with other programming languages (scripting, Python and others) allows building systems for automatic typesetting, integration with databases, ... We can manage scientific projects (Physics, Mathematics, ...) in languages such as Spanish, English, German and Basque.

Veytsman, Boris

132 Warbler Ln.
 Brisbane, CA 94005
 +1 703 915-2406
 Email: borisv (at) lk.net
 Web: <http://www.borisv.lk.net>

T_EX and L^AT_EX consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom L^AT_EX packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in T_EX and three decades of experience in teaching & training. I have authored more than forty packages on CTAN as well as Perl packages on CPAN and R packages on CRAN, published papers in T_EX-related journals, and conducted several workshops on T_EX and related subjects. Among my customers have been Google, US Treasury, FAO UN, Israel Journal of Mathematics, Annals of Mathematics, Res Philosophica, Philosophers' Imprint, No Starch Press, US Army Corps of Engineers, ACM, and many others.

We recently expanded our staff and operations to provide copy-editing, cleaning and troubleshooting of T_EX manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

Webley, Jonathan

Flat 11, 10 Mavisbank Gardens
 Glasgow, G1 1HG, UK
 07914344479

Email: [jonathan.webley \(at\) gmail.com](mailto:jonathan.webley@gmail.com)

I'm a proofreader, copy-editor, and L^AT_EX typesetter. I specialize in math, physics, and IT. However, I'm comfortable with most other science, engineering and technical material and I'm willing to undertake most L^AT_EX work. I'm good with equations and tricky tables, and converting a Word document to L^AT_EX. I've done hundreds of papers for journals over the years. Samples of work can be supplied on request.

TUG 2019
Palo Alto, California
USA
August 9–11, 2019
tug.org/tug2019

13th ConT_EXt Meeting
Bassenge
Belgium
Sept. 16–21, 2019
gust.org.pl/bachotex

TUG financial statements for 2018

Karl Berry, TUG treasurer

The financial statements for 2018 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: <https://tug.org/tax-exempt>.

Revenue (income) highlights

Membership dues revenue was slightly up in 2018 compared to 2017; this was the first year of offering trial memberships, and we ended the year with 1,214 members. Contributions were up about \$2,000, and product sales (mainly Lucida) were down about the same. We had no extra services income. Other categories were about the same. Overall, 2018 income was down < 1%.

Cost of Goods Sold and Expenses highlights, and the bottom line

TUGboat production cost was down substantially, due mostly to printing fewer pages. Other cost categories were about the same. The bottom line for 2018 was strongly negative, about \$10,500, though still a substantial improvement over 2017.

Balance sheet highlights

TUG's end-of-year asset total is down by around \$8,040 (4.4%) in 2018 compared to 2017, following the bottom-line loss.

Committed Funds are reserved for designated projects: L^AT_EX, CTAN, the T_EX development fund, and others (<https://tug.org/donate>). Incoming donations are allocated accordingly and disbursed as the projects progress. TUG charges no overhead for administering these funds.

The 2018 Conference number is the net of the TUG'18 conference (a gain of about \$2000), the loss (about \$1100) from the cancelled PracT_EX'18, and preliminary TUG'19 registrations and expenses.

The Prepaid Member Income category is member dues that were paid in earlier years for the current year (and beyond). The 2018 portion of this liability was converted into regular Membership Dues in January of 2018. The payroll liabilities are for 2018 state and federal taxes due January 15, 2019.

Summary

Due to the trial membership initiative, we ended 2018 with 36 more members than in 2017. We hope the positive trend continues.

Additional ideas for attracting members, or benefits TUG could provide, would be very welcome.

TUG 12/31/2018 (vs. 2017) Revenue, Expense

	<u>Dec 31, 18</u>	<u>Dec 31, 17</u>
ORDINARY INCOME/EXPENSE		
Income		
Membership Dues	77,825	76,502
Product Sales	3,672	7,100
Contributions Income	9,463	7,654
Interest Income	870	546
Advertising Income	270	315
Services Income	<u>761</u>	<u>761</u>
Total Income	92,101	92,879
Cost of Goods Sold		
TUGboat Prod/Mailing	(17,410)	(23,677)
Software Prod/Mailing	(2,550)	(2,599)
Members Postage/Delivery	(1,470)	(2,901)
Lucida Sales to B&H	(1,465)	(2,895)
Member Renewal	<u>(317)</u>	<u>(364)</u>
Total COGS	<u>(23,211)</u>	<u>(32,437)</u>
Gross Profit	68,890	60,442
Expense		
Contributions made by TUG	(2,000)	(2,000)
Office Overhead	(14,301)	(13,741)
Payroll Expense	(63,078)	(63,186)
Professional Fees	<u>(38)</u>	<u>(38)</u>
Interest Expense	(4)	(45)
Total Expense	<u>(79,383)</u>	<u>(79,011)</u>
Net Ordinary Income	(10,493)	(18,568)
OTHER INCOME/EXPENSE		
Prior year adjust	<u> </u>	(3,356)
Net Other Income	<u> </u>	(3,356)
NET INCOME	<u>(10,493)</u>	<u>(15,212)</u>

TUG 12/31/2018 (vs. 2017) Balance Sheet

	<u>Dec 31, 18</u>	<u>Dec 31, 17</u>
ASSETS		
Current Assets		
Total Checking/Savings	176,530	184,765
Accounts Receivable	<u>470</u>	<u>275</u>
Total Current Assets	177,000	185,040
LIABILITIES & EQUITY		
Current Liabilities		
Committed Funds	44,442	42,971
TUG Conference	1,000	596
Administrative Services	2,698	2,698
Prepaid Member Income	6,375	6,070
Payroll Liabilities	<u>1,353</u>	<u>1,080</u>
Total Current Liabilities	55,869	53,416
Equity		
Unrestricted	131,624	146,836
Net Income	<u>(10,493)</u>	<u>(15,212)</u>
Total Equity	121,131	131,624
TOTAL LIABILITIES & EQUITY	<u>177,000</u>	<u>185,040</u>

TUG Business

TUG 2019 election

Nominations for TUG President and the Board of Directors in 2019 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 is no requirement for a ballot this election.

For President, Boris Veytsman was nominated. As there were no other nominees, he is duly elected and will serve for a two-year term.

For the Board of Directors, the following individuals were nominated:

Barbara Beeton, Jim Hefferon, Norbert Preining. As there were not more nominations than open positions, all the nominees are duly elected to a four-year term. Thanks to all for their willingness to serve.

Terms for both President and members of the Board of Directors will begin at the Annual Meeting.

Board members Susan DeMeritt, Michael Doob, and Cheryl Ponchin have decided to step down at the end of this term. All have been TUG board members for many years, and their dedication and service to the community are gratefully acknowledged.

Election statements by all candidates are given below. They are also available online, along with announcements and results of previous elections.

- ◇ Karl Berry
for the Elections Committee
tug.org/election

Boris Veytsman



(Candidate for TUG President.)

I was born in 1964 in Odessa, Ukraine and have a degree in Theoretical Physics. I am a Principal Research Scientist with Chan Zuckerberg Initiative and an adjunct professor at George Mason University. I also do T_EX consulting for a number of customers from publishers to universities to government agencies to non-profits. My CV is at <http://borisv.lk.net/cv/cv.html>.

I have been using T_EX since 1994 and have been a T_EX consultant since 2005. I have published a

number of packages on CTAN and papers in *TUGboat*. I have been a Board member since 2010, Vice-President since 2016, and President since 2017. I am an Associate Editor of *TUGboat* and support tug.org/books/.

During the last two years I have been working with the Board as TUG President on keeping TUG relevant for the user community. I hope to continue this work if elected.

We performed experiments with membership options (lotteries for new members, trial membership) which helped to alleviate the steady loss of individual members over the years. I think we need to increase the effort to recruit institutional members.

I have spent some time working on the Web pages for Education and Accessibility Working groups. We have had a very interesting accessibility workshop at TUG'18 in Rio. There is a pilot project (joint with GUST) of sponsoring participation of Ukrainian students in BachoTeX planned for 2019.

We increased our outreach effort and established a presence in Facebook and Twitter (I am grateful to Norbert Preining for the curation of our FB pages).

I consider our flagship journal to be one of the main activities of TUG. We solicited interesting papers for *TUGboat*, and the recent issues are, in my opinion, quite good. I hope our shortening of embargo period for *TUGboat* will make the journal even more important and relevant.

I think we need to do more outreach to the T_EXers outside TUG, including the huge audience of Overleaf users, students, institutions and many others. This is probably one of the most important tasks for the TUG officers.

I hope the community will give me the honor of serving TUG for another term.

Barbara Beeton



(Candidate for TUG Board of Directors.)

Biography: For T_EX and the T_EX Users Group:

- charter member of the T_EX Users Group; charter member of the TUG Board of Directors;
- *TUGboat* production staff since 1980, Editor since 1983;
- Don Knuth's "T_EX entomologist", i.e., bug collector, through 2014;

- TUG committees: publications, bylaws, elections;
- liaison from Board to Knuth Scholarship Committee 1991–1992.

Retiring from the American Mathematical Society in February 2019.

- Staff Specialist for Composition Systems; involved with typesetting of mathematical texts since 1973; assisted in initial installation of \TeX at AMS in 1979; implemented the first AMS document styles; created the map and ligature structure for AMS cyrillic fonts.
- Standards organizations: active 1986–1997 in: ANSI X3V1 (Text processing: Office & publishing systems), ISO/IEC JTC1/SC18/WG8 (Document description and processing languages); developing the standard ISO/IEC 9541:1991 Information technology — Font information interchange.
- AFII (Association for Font Information Interchange): Board of Directors, Secretary 1988–1996.
- STIX representative to the Unicode Technical Committee for adoption of additional math symbols, 1998–2012, with continuing informal connections.

Statement: Although I will be retired from the AMS I intend to continue to be active in TUG, where I have made so many good friends. As the oldest user group in the worldwide \TeX community, TUG provides a focus for dedicated \TeX users and developers.

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.

Jim Hefferon



(Candidate for TUG Board of Directors.)

Experience: I have used \TeX and \LaTeX since the early 90s for a variety of projects, including technical books and articles. I helped run CTAN for a decade. I have been on the TUG Board a number of terms, including terms as Vice President, and I had the privilege of acting as President for a short time.

Goals: Bringing in new users and keeping them is a constant challenge. This includes both members

of the \TeX community in general and members of TUG. I am particularly interested in helping beginners, who come with challenges of their own. It is sometimes possible for a group of experienced users to forget them and I hope I can help keep them in focus.

Norbert Preining



(Candidate for TUG Board of Directors.)

Biography: I am a mathematician and computer scientist living and working wherever I find a job. After my studies at the Vienna University of Technology, I moved to Tuscany, Italy, for a Marie Curie Fellowship. After another intermezzo in Vienna I have settled in Japan since 2009, first working at the Japan Advanced Institute of Science and Technology, now for Accelia Inc., a CDN/IT company in Tokyo.

After years of being a simple user of (\LaTeX) , I first started contributing to \TeX Live by compiling some binaries in 2001. In 2005, I started working on packaging \TeX Live for Debian, which has developed into the standard \TeX package for Debian and its derivatives. During EuroBach \TeX 2007, I got (by chance) involved in the development of \TeX Live itself, which is now the core of my contribution to the \TeX world. Up till now I am continuing with both these efforts.

Furthermore, with my move to Japan I got interested in its typographic tradition and support in \TeX . I am working with the local \TeX users to improve overall Japanese support in \TeX (Live). In this course we managed to bring the TUG 2013 conference for the first time to Japan.

More details concerning my involvement in \TeX , and lots of anecdotes, can be found at the TUG interview corner (tug.org/interviews/preining.html) or on my web site (preining.info).

Statement: After many years in the active development and four years on the board of directors of TUG, I want to continue serving TUG and the wider \TeX community.

The challenges I see for TUG remain the same over the years: increase of members and funds, and technical improvement of our software. Promoting \TeX as a publishing tool also outside the usual math/CS environment will increase the acceptance of \TeX , and by this will hopefully bring more members to TUG.

Calendar

2019

- | | |
|---|---|
| <p>May 1–5 BachoT_EX 2019, “T_EX old but gold: the durability of typographic T_EXnologies”, 27th BachoT_EX Conference, Bachotek, Poland.
www.gust.org.pl/bachotex/2019-en</p> <p>May 15 TUG 2019 deadline for abstracts for presentation proposals.
tug.org/tug2019</p> <p>Jun 1 TUG 2019 early bird registration deadline. tug.org/tug2019</p> <p>Jun 2–7 18th Annual Book History Workshop, Texas A & M University, College Station, Texas.
library.tamu.edu/book-history</p> <p>Jun 7–14 Mills College Summer Institute for Book and Print Technologies, Oakland, California.
millsbookartsummer.org</p> <p>Jun 10–
Jul 13 Type Paris 2019, intensive type design program, Paris, France. typeparis.com</p> <p>Jun 19–21 The 7th International Conference on Typography and Visual Communication (ICTVC), “Challenging Design Paths”, Patras, Greece. www.ictvc.org</p> <p>Jun 24–27 Book history workshop, Institut d’histoire du livre, Lyon, France. ihl.enssib.fr</p> <p>Jul 3–5 Seventeenth International Conference on New Directions in the Humanities (formerly Books, Publishing, and Libraries), “The World 4.0: Convergences of Knowledges and Machines”, University of Granada, Granada, Spain.
thehumanities.com/2019-conference</p> <p>Jul 4–5 International Society for the History and Theory of Intellectual Property (ISHTIP), 11th Annual Workshop, “Intellectual Property and the Visual”. Sydney, Australia.
www.ishtip.org/?p=995</p> <p>Jul 9 TUG 2019 hotel reservation discount deadline. tug.org/tug2019</p> | <p>Jul 9–12 Digital Humanities 2019, Alliance of Digital Humanities Organizations, Utrecht, The Netherlands.
adho.org/conference</p> <p>Jul 15–19 SHARP 2019, “Indigeneity, Nationhood, and Migrations of the Book”. Society for the History of Authorship, Reading & Publishing. University of Massachusetts, Amherst, Massachusetts. www.sharp2019.com</p> <p>Jul 28–
Aug 1 SIGGRAPH 2019, “Thrive together”, Los Angeles, California.
s2019.siggraph.org</p> <p>Jul 29–
Aug 2 Balisage: The Markup Conference, Rockville, Maryland. www.balisage.net</p> <hr/> <p>TUG 2019 Palo Alto, California</p> <p>Aug 8 Opening reception, 4:00–6:00 PM</p> <p>Aug 9–11 The 40th annual meeting of the T_EX Users Group.
tug.org/tug2019</p> <hr/> <p>Aug 18 <i>TUGboat</i> 40:2 (proceedings issue), submission deadline.</p> <p>Aug 28–
Sep 1 TypeCon 2019, Minneapolis, Minnesota.
typecon.com</p> <p>Sep 4–7 Association Typographique Internationale (ATypI) annual conference, Tokyo, Japan. atypi2019.dryfta.com</p> <p>Sep 15–20 XML Summer School, St Edmund Hall, Oxford University, Oxford, UK.
xmlsummerschool.com</p> <p>Sep 16–21 13th International ConT_EXt Meeting, “Dirty tricks & dangerous bends”, Bassenge, Belgium.
meeting.contextgarden.net/2019</p> <p>Sep 23–26 19th ACM Symposium on Document Engineering, Berlin, Germany.
www.documentengineering.org/doceng2019</p> <p>Oct 26 GuIT Meeting 2019, XVI Annual Conference, Turin, Italy.
www.guitex.org/home/en/meeting</p> |
|---|---|

Status as of 15 April 2019

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568, email: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

User group meeting announcements are posted at tug.org/meetings.html. Interested users can subscribe and/or post to the related mailing list, and are encouraged to do so.

Other calendars of typographic interest are linked from tug.org/calendar.html.

Introductory

- 4 *Barbara Beeton* / Editorial comments
 - typography and *TUGboat* news
- 5 *Sarah Lang* and *Astrid Schmörlzer* / Noob to Ninja: The challenge of taking beginners' needs into account when teaching L^AT_EX
 - problems, solutions, and a manifesto to help all new users
- 3 *Boris Veytsman* / From the president
 - thoughts on changes, fast and slow

Intermediate

- 89 *Karl Berry* / The treasure chest
 - new CTAN packages, October 2018–April 2019
- 17 *Susan Jolly* / Nemeth braille math and L^AT_EX source as braille
 - braille examples of math vs. L^AT_EX source; software project guide
- 28 *Siep Kroonenberg* / New front ends for T_EX Live
 - standalone Tcl/Tk-based GUIs for the TL installer and `tlshell` for `tlmgr`
- 44 *L^AT_EX Project Team* / L^AT_EX news, issue 29, December 2018
- 10 *Carla Maggi* / The DuckBoat — News from T_EX.SE: Processing text files to get L^AT_EX tables
 - errata, Cinderella topics, using `csvsimple` to produce tables from data
- 14 *Mike Roberts* / No hands — the dictation of L^AT_EX
 - `mathfly.org` add-on to Dragon speech recognition for math
- 69 *Joseph Wright* / Real number calculations in L^AT_EX: Packages
 - user-level comparison of floating point arithmetic packages, recommending `xfp`
- 30 *Yihui Xie* / TinyTeX: A lightweight, cross-platform, and easy-to-maintain L^AT_EX distribution based on T_EX Live
 - on-the-fly T_EX Live package installation for R users

Intermediate Plus

- 82 *Karl Berry* / *TUGboat* online, reimplemented
 - (re)generation of *TUGboat* tables of contents and accumulated lists
- 22 *Jim Fowler* / Both T_EX and DVI viewers inside the web browser
 - compiling Pascal to WebAssembly to run ϵ -T_EX in a browser, including `TikZ`
- 34 *Hans Hagen* / ConT_EXt LMTX
 - LuaMetaT_EX, a minimized engine for future ConT_EXt and other experiments
- 76 *Michal Hoftich* / T_EX4ht: L^AT_EX to Web publishing
 - configuring HTML, XML, . . . generation with T_EX4ht and `make4ht`
- 25 *Vít Novotný* / Markdown 2.7.0: Towards lightweight markup in T_EX
 - a Lua command-line interface, `doc`, and content slicing support for `markdown`
- 47 *Nicola Talbot* / Glossaries with `bib2gls`
 - indexing infrastructure with `glossaries-extra` and/or using `.bib` format
- 71 *Joseph Wright* / Real number calculations in T_EX: Implementations and performance
 - precision, accuracy, and performance comparison of existing floating point arithmetic packages
- 33 *Joseph Wright* / Extending primitive coverage across engines
 - `\expanded` and other primitives available in pdfT_EX, X_YL_AT_EX, (u)pT_EX

Advanced

- 61 *Enrico Gregorio* / T_EX.StackExchange cherry picking, part 2: Templating
 - extended `expl3` programming examples: templating and double loops, ISBN, catcode tables, and more
- 38 *Khaled Hosny* / Bringing world scripts to LuaT_EX: The HarfBuzz experiment
 - extended discussion of using HarfBuzz to shape text for LuaT_EX output

Reports and notices

- 2 Institutional members
- 91 *John Atkinson* / Comic: Punctuation headlines
- 85 *Boris Veytsman* / Book review: *Never use Futura* by Douglas Thomas
 - review of this wide-ranging book on the history and design of Futura
- 88 From other T_EX journals: *Die T_EXnische Komödie* 1/2019; *Zpravodaj* 2018/1–4
- 91 T_EX consulting and production services
- 93 *Karl Berry* / TUG financial statements for 2018
- 94 *TUG Elections committee* / TUG 2019 election
- 96 Calendar