TUGboat

Volume 43, Number 1 / 2022

General Delivery
3 From the president / Boris Veytsman
4 Editorial comments / Barbara Beeton
   Robin Fairbairns, 1947–2022;
   R.I.P. Brent Longborough;
   TUG 2022 — Online again;
   HTML versions of \TeX-related documents;
   Movable metal type;
   Calling all users of the UK \TeX FAQ
6 Robin Fairbairns and UK TUG / Jonathan Fine
7 The last decade at GUTenberg / Jacques André, Patrick Bideault, Denis Bitouzé, Michel Bovani, Thierry Bouche, Maxime Chupin, Daniel Flipo, Yvon Henel

\LaTeX
10 Markdown 2.15.0: What’s new? / Vít Novotný, Dominik Rehák, Michal Hoftich, Tereza Vrabcová

Tutorials
16 The DuckBoat — Beginners’ Pond: CDs, but not Compact Disks / Carla Maggi

Graphics
23 Making open source textbooks, and diagrams with AlDraTeX / Seth Bergmann

Methods
28 Automatically removing widows and orphans with \texttt{lua-widow-control} / Max Chernoff

Software & Tools
40 l3build: The beginner’s guide / Joseph Wright
44 \texttt{bib2gls}: standalone entries and repeated lists (a little book of poisons) / Nicola Talbot

Macros
59 Transparent file I/O using the original \TeX program and the plain \TeX format / Udo Wermuth

Reviews
73 Book review: \texttt{\LaTeX} Beginner’s Guide, second edition, by Stefan Kottwitz / Sarah Lang

Hints & Tricks
75 The treasure chest / Karl Berry
77 \TeX Live 2022 news / Karl Berry

Abstracts
78 \texttt{Die \TeXnische Komödie}: Contents of issues 4/2021–1/2022
79 \texttt{Zpravodaj}: Contents of issue 2021/1–4

TUG Business
2 TUGboat editorial information
2 TUG institutional members
80 TUG Annual General Meeting Procedures
81 TUG financial statements for 2021 / Karl Berry

Advertisements
82 \TeX consulting and production services
83 \TeXnology Inc.

News
84 Calendar
Individual memberships
2022 dues for individual members are as follows:
  - Trial rate for new members: $30.
  - 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 \textit{TUGboat} and other benefits electronically, at a discount. All membership options are described at \url{tug.org/join}.

Membership in the \TeX{} Users Group is for the calendar year, and includes all issues of \textit{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
\textit{TUGboat} subscriptions (non-voting) are available to libraries and other organizations or individuals for whom memberships are either not appropriate or desired. Subscriptions are delivered on a calendar year basis. The subscription rate for 2022 is $115.

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

Trademarks
Many trademarked names appear in the pages of \textit{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.

Board of Directors
Donald Knuth, \textit{Ur Wizard of T\TeX{}-arcana}†
Boris Veytsman, \textit{President}∗
Arthur Rosendahl∗, \textit{Vice President}
Karl Berry∗, \textit{Treasurer}
Klaus Höppner∗, \textit{Secretary}
Barbara Beeton
Johannes Braams
Paulo Cereda
Kaja Christiansen
Ulrike Fischer
Jim Hefferon
Frank Mittelbach
Ross Moore
Norbert Preining
Raymond Goucher (1937–2019),
\textit{Founding Executive Director}
Hermann Zapf (1918–2015), \textit{Wizard of Fonts}
* member of executive committee
† honorary

See \url{tug.org/board} for a roster of all past and present board members, and other official positions.

Addresses
\TeX{} Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Telephone
+1 503 223-9994

Fax
+1 815 301-3568

Web
\url{tug.org}
\url{tug.org/TUGboat}

Electronic mail
General correspondence, membership, subscriptions:
\url{office@tug.org}

Submissions to \textit{TUGboat}, letters to the Editor:
\url{TUGboat@tug.org}

\TeX{}Xnical support, public mailing list:
\url{support@tug.org}

Contact the Board of Directors:
\url{board@tug.org}

Copyright © 2022 \TeX{} 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 \TeX{} Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.
[The author] hopes that nobody will ever scrutinize any of his own writings as meticulously as he and others have examined the ALGOL Report.

Donald E. Knuth
“The Remaining Trouble Spots in ALGOL 60”
*Communications of the ACM*
(Volume 10, Number 10, October 1967)
TUGboat editorial information
This regular issue (Vol. 43, No. 1) is the first issue of the 2022 volume year. The deadline for the second issue in Vol. 43 (the TUG’22 conference proceedings) is July 31, 2022, and for the third (regular) issue is October 15, 2022. Contributions are requested.

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

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 TeX and LaTeX, are available from CTAN and the TUGboat web site, and are included in TeX distributions. We also accept submissions using ConTeXt. For deadlines, templates, author tips, and more, see 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 TeX 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.

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, ams.org
Association for Computing Machinery, New York, New York, acm.org
Aware Software,
Newark, Delaware, awaresw.com
Center for Computing Sciences,
Bowie, Maryland

CSTUG, Praha, Czech Republic,
cstug.cz
CTAN, ctan.org
Duke University Press, Durham,
North Carolina, dukepress.edu
Hindawi Foundation, London, UK,
hindawi.org
Institute for Defense Analyses,
Center for Communications Research,
Princeton, New Jersey
L3Harris, Melbourne, Florida,
l3harris.com
IAmTeX Project, latex-project.org
MacTeX, tug.org/mactex
Maluhy & Co., São Paulo, Brazil,
maluhy.com.br
Marquette University,
Milwaukee, Wisconsin,
marquette.edu
Masaryk University,
Faculty of Informatics,
Brno, Czech Republic, fi.muni.cz
Nagwa Limited, Windsor, UK, nagwa.com
Overleaf, London, UK,
overleaf.com
StackExchange,
New York City, New York,
tex.stackexchange.com
TeXFolio, Trivandrum, India,
texfolio.org
Université Laval, Ste-Foy, Québec,
Canada, bibl.ulaval.ca
University of Ontario, Institute of Technology,
Oshawa, Ontario, Canada, ontariotechu.ca
University of Oslo,
Institute of Informatics,
Blindern, Oslo, Norway, uio.no
VTEx UAB, Vilnius, Lithuania,
vtex.lt
From the president
Boris Veytsman

My previous two columns were written about licenses for free software. The topic is vast, and I would like to discuss another aspect of it today.

First, a little bit of history. Gutenberg’s printing press was a revolutionary invention, which radically decreased the cost of the written word. One of the first applications of the invention was the printing of indulgences. One of the first printed indulgences was presumably produced by Gutenberg himself (Figure 1). Indulgences were one of the reasons for Martin Luther’s revolt against the Catholic church. Luther himself understood the power of the printing press very well. He was able to publish his tracts in hundreds of thousands of copies. Luther hired the best printers, including the famous illustrator Lucas Cranach the Elder. However, Luther’s adversaries, both Catholics and competing Protestants, also used the press to spread their texts. Some historians say that the Reformation and Counter-Reformation would not be possible without Gutenberg’s invention.

This leads to an interesting question. Religious wars took millions of lives in the 16th and 17th centuries. Did the widespread use of incendiary pamphlets contribute to this carnage? And if yes, how much blame can we put on the printing press?

Similar things occurred many times. It is common knowledge that most human inventions were quickly used to kill human beings. The domestication of the horse gave rise to cavalry. The invention of the wheel led to military chariots, people learned to fly and started to bomb other people, and so forth. What I would like to stress here is that information technologies are not an exception to this rule. From writing to printing press to telegraph to radio to television to computers—every innovation was quickly used for killing people, or for military propaganda, which also led to killing people. Even our \TeX\ work may have military uses. I must confess that in my \TeX\ portfolio there are styles written and paid for by the US Army.

Thus those of us who work in information technologies can be reasonably sure that our work will someday be used to kill. A fresh example: after the vendors of commercial operating systems left Russia, it plans to switch its military to the “Russian OS”, reportedly being a clone of Linux. Can Free Software prevent this? A naïve approach would be to try to add the clause “Thou shalt not kill with this software” to the license. However, free software guidelines such as DFSG explicitly forbid discrimination with respect to the field of use. If you forbid the use of your software for nefarious aims, you automatically make your software non-free. Are these guidelines reasonable? I think yes. At the end of the day the rule of law is the moral authority. If somebody wants to use your software for amoral purposes, they can easily disregard this clause of your license. On the other hand, such clause may prevent other people from using your work for defense. There was no way to prevent the tables of logarithms, sines and cosines to be used in artillery: once a mathematical theorem is published, it can be used by anybody for any purpose.

I think I can agree with the position of Canonical, the maintainer of Ubuntu (ubuntu.com/blog/canonical-stance-with-ukraine). In response to the Russian invasion of Ukraine, Canonical terminated support, professional services and partnership with Russian users. However, it did not restrict the access for security patches for these users, noting that “free software platforms like Ubuntu, VPN technologies, and Tor, are important for those who seek news and dialogue outside state control”, and directing any subscription income to Ukrainian humanitarian causes.

There are, however, grounds for hope. Besides being used for war propaganda, the printing press started rapid progress of science and literature, that, after all, made us live longer, healthier, and maybe even happier lives. We also can hope that our work at the end of the day will make the world slightly better. We cannot prevent bad people from using our work, but we can hope our work is used by good people.

⋄ Boris Veytsman
president (at) tug dot org

Figure 1: Fragment of 31-line indulgence, from en.wikipedia.org/wiki/31-line_Indulgence
Editorial comments
Barbara Beeton

Robin Fairbairns, 1947–2022

Robin left us on 25 February 2022, after a long period of ill health. An interview by Dave Walden in February 2005\(^1\) tells how he came to the \TeX\ world, even before he joined the Computer Laboratory of the University of Cambridge, bringing (L)\TeX\ with him.

At Cambridge, Robin was in charge of many activities related to \TeX. Early on, he was an avid \TeX\ user, and actively answered questions on texhax and comp.text.tex. When the University of Aston determined that they were no longer able to host the UK\TeX\ Archive, Robin rescued it and brought it to Cambridge, where it became one of the original nodes of \CTAN. (In turn, this freed Sebastian Rahtz to put together the first versions of \TeX\ Live.\(^2\)) Robin is listed as (co-)author and/or maintainer of a rather large number of packages on \CTAN,\(^3\) among them \texttt{capt-of}, \texttt{covington} (linguistic support), \texttt{endnotes}, \texttt{footmisc}, \texttt{moreverb}, \texttt{rotating}, and \texttt{setspace}, to mention some that I’ve seen still in use recently.

Always helpful, and a good explainer, Robin (with others) selected the 100 most interesting questions asked on various Internet fora and created the UK\TeX\ FAQ, which they published in \textit{Baskerville} (The Annals of the UK \TeX\ Users’ Group) in December 1994 (vol. 4, no. 6).\(^4\) The online version moved to Cambridge in 1999, joining \CTAN, with Robin as editor. With the maintenance of \CTAN centralized under DANTE, the Cambridge site became a mirror, and after Robin’s retirement at the end of December 2014 the FAQ found a new home.\(^5\) However, much of the content still owes its existence and clarity to Robin.

Robin attended the first meeting of the British user group, UK-TUG, and was persuaded to serve on the committee; he also served as chairman, and was the formal publisher of \textit{Baskerville}. He was active in TUG as well, serving on the board in 1996–1997. He attended several TUG annual meetings, and was editor of the proceedings for two of them: 1995 (St. Petersburg, Florida) and 2000 (Wadham College, Oxford, UK). The \textit{TUGboat} production team was formed during the editing of the 1995 proceedings, with Robin as a founding member; he remained a member through 2019. Also a \textit{TUGboat} author,\(^6\) Robin introduced the “New (L)\TeX\2\epsilon TUGboat Macros”,\(^7\) a tutorial for the package he helped to develop, based on his experience with the 1995 proceedings.

Last, but far from least, Robin was a member of the \texttt{\LaTeX3} Project. The memorial on their site includes this recognition:\(^8\)

Robin was a tireless member of the community who helped many people for many years with his kind, friendly, and patient approach. He clearly looked to improve the \TeX\ ecosystem and worked hard to leave it in a better place after his time was up. He certainly succeeded in that.

I couldn’t agree more, and will miss him greatly.

R.I.P. Brent Longborough

Brent Longborough, an active participant since its inception in \TeX\Stackexchange and other \tex\-related pursuits, left us peacefully on 6 December 2021, surrounded by his family. Brent characterized himself as an “old-ish IT geezer, young at heart”.

Born in Exeter, in Devon, England, he moved with his family to a number of different UK towns and cities, prefiguring his later peripatetic life. He attended Oxford University, concentrating in chemistry, but decided early that this was not for him, and switched to IT, which became his permanent direction.

In 1980, he moved to Rio de Janeiro, working for a British airline. There he mastered the Portuguese language, met his wife, Celia Navarro, and acquired a family.

\(^{1}\) tug.org/interviews/fairbairns.html
\(^{2}\) tug.org/TUGboat/tb18-2/tb55tguide2.pdf
\(^{3}\) ctan.org/author/fairbairns
\(^{4}\) mirrors.ctan.org/usergrp\-uits/baskervi/bask4_6.pdf
\(^{5}\) texfaq.org
\(^{6}\) tug.org/TUGboat/Contents/listauthor.html#Fairbairns,Robin
\(^{7}\) tug.org/TUGboat/tb17-3/tb52guide.pdf
\(^{8}\) latex-project.org/news/2022/03/16/robin-fairbairns/
For many years, Brent was employed by IBM, still in Brazil. After leaving IBM, he worked for a Turkish airline, moving with his family to Turkey, becoming conversant in Turkish as well. When he became involved with \TeX, he participated in the development of arara, including the translation of arara into Turkish.

Brent continued moving around the world, living at various times in Mexico, Colombia and India, returning to the UK after retirement, where he settled with his family in Wales. Languages were well represented in Brent’s skill set, and included French and Spanish (as well as innumerable programming languages), and finally, Welsh. After his return to the UK in 2002, Brent served (in 2013) as a member of the UK-TUG committee.

Brent’s contributions to CTAN were related to the use of \git, and the package \texttt{tagging}, which supported the creation of a source document from which multiple distinct outputs could be generated.

A representative example of his contributions to Stackexchange discussions is his response to the question “Why is \LaTeX so complicated?”, which can be seen at \url{https://tex.stackexchange.com/a/222505}.

A memorial to Brent was held via Zoom on 23 January 2022, led by his son, Gus Navarro, and attended by many of his friends. Conducted alternately in Portuguese and English, his message honored his father’s care and attention through an eventful life. Remembrances by friends reinforced the memory of these qualities.

Thanks to Paulo Cereda for sharing notes from the Zoom memorial and providing other helpful information.

**TUG 2022 — Online again**

Owing to the ongoing uncertainty concerning the ability to travel and meet with no restrictions based on COVID-19 status, it has been decided to, once again, conduct the conference online. This will take place from 22–24 July 2022. For details so far visit the conference web page at \url{tug.org/tug2022}.

As always, participation is encouraged, both through submission of papers and volunteering to assist with the organization. Instructions for submitting a presentation are on the web page. If you wish to volunteer, send a message to \url{tug2022@tug.org}.

**HTML versions of \TeX-related documents**

Two significant developments have occurred to support (visual) access to \TeX-related material on the web: an HTML version of the PGF/TikZ manual, and HTML output from articles on the arXiv.

The manual was provided by Dominik Peters at \url{https://tikz.dev}, as announced in January on the \texttt{tex.stackexchange} forum.\footnote{\url{tex.stackexchange.com/a/630330}} It’s searchable, and sections are provided as links identified with ¶. (The graphic of the opening page in the posting is very attractive.)

The second development is the availability of math articles on the arXiv via \url{arxiv.org} as “responsive HTML5 web pages” (the “5” is an indication of the HTML version). Translation from the source is accomplished using \texttt{bT\LaTeXML}.ootnote{\url{math.nist.gov/~BMiller/LaTeXML}} More details of the project can be found on the arXiv blog.ootnote{\url{blog.arxiv.org/2022/02/21/3-ar5iv-articles-as-responsive-web-pages/}} We would be pleased to learn your reactions to the output of this project.

The W3C MathML Working Group is working on the problem of audio and tactile accessibility for visually impaired readers. This is even more complicated than conversion to HTML for sighted readers. At an appropriate time, we hope to learn about the problems they’ve encountered and how those were overcome.

**Movable metal type**

Earlier this year, Boris Veytsman sent to me and several others interested in the printing arts a link to the scans of a beautiful copy of a Gutenberg bible, resident in the Beinecke Rare Book and Manuscript Library at Yale University.ootnote{\url{orbis.library.yale.edu/uuids/holdingsInfo?bibId=3391099, collections.library.yale.edu/catalog/2020598}} Leafing through the pages, I observed that not only was some of the text printed in two colors (black with select passages in red), but ornate initials and delicate marginal decorations were also present. While admiring this handiwork, a followup message arrived from one of the other recipients, admiring the printing craftsmanship that had produced these marvelous decorations.

Stop! Given the mechanical requirements of printing with movable type, it wasn’t possible that the decorations could be produced on the press. They must be added by hand after the print run. Since I would have expected the individual making these comments to know the process, but clearly that was not the case, it seemed appropriate to present a brief introduction. Here I will be very brief, but with the intention of creating something closer to a “how to” manual at a later time.

Movable type, from Gutenberg’s time (mid-15th century) through the middle of the 19th century, consisted of individually cast metal letters that were
composed into text by hand. In order to keep them in their proper locations, it was necessary to place them on a flat solid surface and constrain them by locking them into a frame, or chase, so that they wouldn’t fall apart into “pi”, forcing the compositor to start all over, after picking up the little pieces and sorting them into a recognizable order.

Watch this brief video, showing how an 83-year-old Japanese printer carefully selects her type, arranges it, locks it up on the press, and prints a business card.\textsuperscript{13} Then contemplate how much more effort went into the printing of the Gutenberg bible.

**Calling all users of the UK TeX FAQ**

This is not new news, but may not be known generally, and it can come as a surprise when links pointing to the FAQ come up broken.

The UK TeX FAQ was created under the auspices of UK-TUG. Full publication first occurred in *Baskerville* 4:6, December 1994, as *The New TeX FAQ: Your 100 Questions Answered*.\textsuperscript{14} At that time, the home for the FAQ had moved from the online comp.text.tex newsgroup to Cambridge University, where it came under the tender care of Robin Fairbairns. When Robin retired, the FAQ was cut loose, but the established url maintained independently, and linked to the new location. This has become too expensive, and the academic ID has been terminated. This leaves a large number of links throughout the Internet without a recognizable target. An effort is being made to update links in online forums or on static pages where this is possible. Of course, it is impossible in such places as mail archives, but since the FAQ itself is still alive (if a bit inactive) anyone knowing of a reference that can be fixed is encouraged to fix it.

Here is the formula. The former syntax of the url was this:

\begin{verbatim}
http://www.tex.ac.uk/cgi-bin/texfaq2html?label-\{topic\}
\end{verbatim}

This should now be

\begin{verbatim}
https://texfaq.org/FAQ-\{topic\}
\end{verbatim}

Please use the new form when referring to the FAQ, and if you encounter the old url, please correct it if you can, or ask someone who can.

\textsuperscript{13} youtube.com/watch?v=UqDU6U01328
\textsuperscript{14} http://uk.tug.org/wp-installed-content/uploads/2008/12/46.pdf
The last decade at GUTenberg

Abstract
The French T\TeX user group, GUTenberg, has elected a new board in November 2020. Since then, it is coming back to life and serves again the French-speaking community.

But in the meantime, a letter written by the former President of GUTenberg, Jérémy Just, still a member of the board, was published in TUGboat 42:1, pp. 12–13. Unfortunately, it was highly inaccurate and incomplete, and we are exercising our right to reply to it. (Indeed, we would say more about the character of that letter but it is best to avoid raising our voices on these pages.)

1 Local groups: inception and day-to-day work
A local users' group’s day-to-day activity is made out of little things: replying to emails; updating a website; updating the members’ register; preparing the next issues of the group’s publications; caring for a server; doing some bookkeeping; preparing the next gatherings or training sessions; providing software to members; and so on.

These little things need to be sorted out, one by one, so that, one year after the other, the group goes on. It is a delicate mechanism that needs caring people to attend it.

Back in the days before the web, any local \TeX users’ group, at its inception, was providing software (and information about it) to its members. The groups were lively: joining them was almost the only way to have the chance to use \TeX.

Then our international community of users made the software programs and their documentation available on the internet, so that anyone could have access to it, freely and instantly. This amazing achievement was due to the step-by-step process described above: some passionate users were dedicating some of their time to it.

2 \TeX 24/7 online availability and its effect on LUGs
When this goal was reached, when everything that was needed to use (\LaTeX)\TeX became available 24/7, local users’ groups had to prove themselves to still be useful. Some members who had joined before in order to be provided with software and documentation were not to be seen any more.

A decreasing membership affected the fragile mechanism described above. It became even more fragile. And needed even more care than before. More dedication—at the very moment where its workforce was decreasing. At the very moment where its usefulness was at stake.

It happened here and there. For example in France, at GUTenberg, a decade ago.

Luckily, since then, in most cases, the users’ groups have proved useful: the dedication of their members, their good will, the love for \TeX that they spread around showed that the usefulness of a group goes far beyond providing software and documentation. When sharing them, we share much more than them: at the end, it’s about looks and smiles, just as any human activity.

And that is the fun of it.

3 An agony
Unfortunately, a decade ago, things went on differently at GUTenberg. When more care was needed, care was diminishing. When lack of care was mentioned, dissent emerged—instead of better care.

It is a classic story, made out of classic issues that could have been fixed by people binding other people together. Minor issues that could have been fixed by patience, care and good will.

But surprisingly, in times when communication can be easy, misunderstanding prevailed. Passionate people left, tired of seeing their ideas not being replied to in the way that they would have expected before.

The fragile mechanism described above wasn’t running smoothly any more. The group began to lack people to care for its day-to-day activities. At GUTenberg’s board, more and more people were missing, some of them tired from seeing their projects being rebuffed by a president who was lacking time.

GUTenberg’s activity was harshly decreasing. Its journal was seldom published—before ceasing publication. Some of its subscribers complained by email; some emails were not even replied to. During those hard times, calls for help were heard; some were granted hearing. Some not. GUTenberg’s mandatory annual gatherings were not organized any more.

And it is all about looks and smiles, just as any human activity: when the activity itself is missing, the good will fades out quickly. The president was by then alone. Because of his own inability to work with others. It is a pity to read his words about “the atmosphere among the board is far from the friendly one that we had when the board was working smoothly”: there was no work and no atmosphere at all. This is how a group dies: when only one person remains, you can’t call it a group any more.

No publications. No gatherings. No financial reports sent to the members. No banking activity at all in 2019. A few subscriptions remained, but the...
joint GUTenberg+TUG subscriptions weren’t even reported to TUG. No emails on the board’s mailing list from August to December 2018. These are facts. Not “personal point of views”, such as the ones in the article that we are replying to.

4 Memory matters
The group was dead. But its memory remained. Former members were still around. Still using \TeX. Lonely, but remembering. And some started to talk together. To talk about the group that they were missing: GUTenberg. Its name was mentioned again. And again. Nothing blazing. But embers. And some breath blown on it. By some people still caring about a group that used to be. Each of them concerned about GUTenberg’s unacceptable demise.

We were not many. But we were . . . a group.

5 A call for change
On January 17, 2020, an email was sent to the association’s only member still in charge:

Subject: AG GUTenberg
Date: Fri, 17 Jan 2020 09:44:53 +0100
Hello Jérémy,
We send you this email as a collective; you will find our names hereunder.
We thank you for updating GUTenberg’s website so that online subscriptions are now available for 2020.
We are well aware that you have spent and still spend a lot of energy for the association and that you are overwhelmed by your professional activity.
Our bylaws provide for an annual general assembly with moral and financial reports. Nevertheless, no general assembly was held since 2013, no report was published since then. In addition to that, the board members are only elected for a four-years term, i.e. since 2017, no one has a valid term any more.
It is therefore urgent to organize a general assembly to decide if we dissolve our association or if we restart GUTenberg on a sound basis.
To this end, it is necessary to have an accurate financial report: a clear view on our assets with a history of our bank transactions, but also a list of our debts to our members who subscribed to services that were not provided, such as our Journal, the TUGboat issues etc.
The best would be for you to call this general meeting, in Paris, before the end of March, where you would present the current state of the association.

In the absence of a positive response from you before the end of January, we would have to call this general meeting ourselves.

Looking forward to reading from you, sincerely,

This message was signed by 8 names, now depicted as a “small group of people”. Such a depiction is a common way to make people suspicious. Well, let’s introduce those people. By alphabetical order:

Jacques André founder, board member 1988–2007; typography historian. Jacques has contributed often to the association’s publications; his books have been reviewed in TUGboat, where he has also published articles.

Patrick Bideault co-opted board member, deputy assistant secretary in charge of the memberships paid by PayPal, now president and maintainer of the coffeeestains package, texnique.fr moderator.

Denis Bitouzé board member 2010–2014, maintainer of several packages, co-creator and moderator of texnique.fr, organizer of the annual (and famous) Dunkirk \LaTeX training, co-author of books about \LaTeX, author of an open access \LaTeX course and tireless \Knex flag-bearer, now secretary.

Thierry Bouche board member 1997–2017, former editor of Cahiers GUTenberg.

Michel Bovani board member 1999–2003, maintainer of the fourier package.

Maxime Chupin board member 2009–2017, maintainer of the bclogo, luamesh and matapli packages, now deputy secretary.


Yvon Henel board member 2009–2017, maintainer of many packages, now deputy treasurer.

We sincerely hope that any suspicion has now vanished. We were only a group of honest people, worried about an association that we had in common, to which we were deeply attached.1 Please notice that we never asked to dissolve GUTenberg. We only mentioned that it could be possible, just like Jérémy Just himself did before in several messages; this mention was due to the association’s by-then current state.

Then, the horrendous article that we are replying to mentions a shadow cabinet. Yes, we have used

1 Four among us have applied to the election and are now in charge at GUTenberg.
this expression once. As a joke. And we deeply regret it: to be understood, a joke needs views to be shared by the addressee. But anyone can understand that we have imagined ourselves in charge of GUTenberg. This is why we have published our election platform, which is not even mentioned in Mr Just’s text: he prefers mentioning imaginary projects blocked on purpose and the abovementioned nonexistent friendly atmosphere among the by-then nonexistent board.

We recognize that Jérémy Just has spent time for the association. However, this time spent has slowly but surely led to the demise of the association. He has gradually disgusted many of us from working with him over the past few years.

As a matter of fact, Jérémy Just does not have the same conception of the association’s work as we do: for him, it is mainly the sum of individual activities (his moral assessment of the General Assembly of 2020 was only an assessment of his T\TeXian activities), whereas for us, it is a collective project, with collective choices, and collective stimulation giving body to the association and making it much more than the sum of the activities of its members.

6 GUTenberg’s revival

To relaunch the association, for the General Assembly that we forced to be held, some of us put together a team, wrote a project and presented ourselves for the election of the board.\footnote{See our election platform (in French) at: gutenberg-asso.fr/IMG/pdf/liste-gut-renouveau--profession-de-fol-projets-equipe.pdf} Their purpose was to rebuild, as a team, and we were glad to be joined by others at the new board, after the election. Nowadays the association serves its community again. But, to our great disappointment, one board member spits in the soup, as we say in French. It is hard for us to understand the acrimonious article we are replying to: why isn’t it its author happy to have new colleagues in the board? Why isn’t he happy to see the association’s revival? It’s one thing to be overwhelmed by a job that you have applied for without being suited for it (and no one blames anyone for this, as it is very hard to know in advance what it means to serve an association). It’s another thing to blame others when they dedicate themselves to clean up the mess and rebuild the house. This is why we have exercised our right of reply.

We apologize to the readers that are not concerned by the matter. But fake news has to be fought. Always.

GUTenberg is back to life: 4 new issues of its bulletin have been published (see \textit{TUGboat} 42:3, p. 313–314), its journal is relaunched (see \textit{TUGboat} 42:3, p. 315), it is rewriting its bylaws, has a new server and a new url, \texttt{gutenberg-asso.fr}, selected by a poll among its members. Of course, it could do more and better, but at least, it works, and it works collectively.

The French-speaking community is lively, and its T\TeX user group is happy to be serving it again. Long live GUTenberg! Long live TUG and LUGs all over the world! Support your community: join your local TUG!

\begin{itemize}
  \item Jacques André  
  Founder, GUTenberg board member  
  1988–2007
  \item Patrick Bideault  
  President 2020–2022 at GUTenberg  
  \texttt{pb-latex (at)gmx dot fr}
  \item Denis Bitouzé  
  Secretary 2020–2022 at GUTenberg
  \item Michel Bovani  
  GUTenberg board member 1999–2003
  \item Thierry Bouche  
  Former editor-in-chief, \textit{Cahiers GUTenberg},  
  board member 1997–2017
  \item Maxime Chupin  
  Deputy secretary 2020–2022 at GUTenberg
  \item Daniel Flipo  
  GUTenberg board member 1993–2005
  \item Yvon Henel  
  Deputy treasurer 2020–2022 at GUTenberg
\end{itemize}
Markdown 2.15.0: What’s new?

Vít Novotný, Dominík Reháč, Michal Hoftich, Tereza Vrabcová

Abstract

At TUG 2021, we celebrated the fifth birthday of Markdown in \TeX. In this article, we introduce new features developed in the months since, and ideas for the future development of the Markdown package.

The article is divided into three sections. In the first two sections, we introduce the new features of Markdown to the two main audiences of Markdown:

1. the writers, who type content in Markdown, and
2. the coders, who prepare templates and solutions.

In Section 3, we discuss ideas for the future to the third audience of Markdown: the developers, who alter and further improve the Markdown package.

1 Writer’s newsletter

Michael Thompson from the pandoc-discuss mailing list characterized Markdown as a perfectly minimalist markup language that only faces the writer with one question: what the next sentence should be. [4] However, for some types of documents, the few structural elements of Markdown can be too few.

The writers may enable the hybrid option and combine \TeX and Markdown markup, but this tends to reduce clarity, stability, and ease of reuse. To reduce the need for hybrid markup, we introduce new syntax extensions for Markdown in sections 1.1–1.4.

Since version 2.10.0 of the Markdown package, writers have been able to redesign their Markdown documents without programming using \LaTeX themes [3]. However, few \LaTeX themes have been publicly available until recently. In Section 1.5, we introduce \LaTeX themes, which self-publishers can use for typesetting books and publishing collaterals.

1.1 Task lists

To track progress on your goals, it can be useful to add checkboxes to list items. Since version 2.11.0, Markdown has supported the taskLists option:

\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}
Tasks:
2. [ ] Draft outline.
3. [ ] Copy edit.
\end{markdown}
\end{document}

Output:

- Tasks:
  - Draft title.
  - Draft outline.
  - Copy edit.

1 github.com/witiko/markdown/issues/95

1.2 Emphatic line breaks

In poems and plays, line breaks carry a meaning and must be preserved. In Markdown, you can write a line break by ending a line with two or more spaces:

Memory and desire, stirring
Dull roots with spring rain.

However, this can be tedious for longer texts. Furthermore, the Markdown package only supports line breaks in the \markdownInput command, because \TeX strips trailing newlines from the input:

\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}
Memory and desire, stirring 
Dull roots with spring rain.
\end{markdown}
\end{document}

Output:

Memory and desire, stirring
Dull roots with spring rain.

Since version 2.12.0, the Markdown package supports the hardLineBreaks option, which makes every line break emphatic:

\documentclass{article}
\usepackage{hardLineBreaks}
\begin{document}
\begin{markdown}
Memory and desire, stirring
Dull roots with spring rain.
\end{markdown}
\end{document}

Output:

Memory and desire, stirring
Dull roots with spring rain.

This makes it easier to typeset long poems and plays.

1.3 Cross-references

In technical and academic writing, cross-references between sections are common. Previously, writers would need to combine \TeX and Markdown markup:

\documentclass{article}
\usepackage{hybrid}
\begin{document}
Conclusion \label{sec:conclusion} ==
In this paper, we have discovered that most grandmas would rather eat dinner with their grandchildren than get eaten. Begone, wolf!
\end{document}

Conclusion \ref{sec:conclusion} ==

2 This limitation of \TeX does not apply to Con\TeXt MkIV; see also github.com/witiko/markdown/issues/101.
3 github.com/witiko/markdown/issues/98
Since version 2.14.0, Markdown has supported attributes on section headings and the `relativeLinks` option,\(^4\) which enables cross-references in Markdown:

```
\begin{markdown}
\documentclass{article}
\usepackage[headerAttributes, relativeLinks]
{markdown}
I conclude in Section `<#sec:conclusion>`.
\end{document}
```

Conclusion `{#sec:conclusion}`

In this paper, we have discovered that most grandmas would rather eat dinner with their grandchildren than get eaten. Begone, wolf!

```
\begin{markdown}
\begin{document}
\maketitle
\begin{markdown}
When Little Red Cap entered the woods a wolf came up to her.
\end{markdown}
\end{document}
```

\section*{1.4 Document metadata}

Even though writers can prepare their documents in Markdown, they previously needed to specify metadata for their documents (such as the title or the author's name) in \TeX: \(\text{\footnotesize \begin{verbatim}
\documentclass{article}
\usepackage{markdown}
<title>{On \textbf{Wolves} & \textit{Grandmas}}</title>
\author{Little Red Cap}
\begin{document}
\maketitle
\begin{markdown}
When Little Red Cap entered the woods a wolf came up to her.
\end{markdown}
\end{document}
\end{verbatim}}\text{\footnotesize}\)

Since version 2.11.0, the Markdown package has supported the `jekyllData` option,\(^5\) which allows us to write metadata in Markdown:

```
\begin{markdown}
\begin{document}
\begin{markdown}
---
title: Of *Wolves* & _Grandmas_
author: Little Red Cap
---
When Little Red Cap entered the woods a wolf came up to her.
\end{markdown}
\end{document}
```

\section*{1.5 \LaTeX themes for self-publishers}

Writers who are unaccustomed to \TeX may find their precious time slipping away, spent scouring online forums looking for a fix for that one thing that is messing up the whole layout. In cooperation with the Writersglen publishing house, we have created a set of \LaTeX themes for typesetting books and publishing collaterals in Markdown.\(^6\)

Let's show the ease of use of these templates with an example. Using the business card template, we might end up with a content file looking like this:

```
# Nemo Green
## Tow Boat Enthusiast
- 1-800 PDF\LaTeX
- tug@boats.com
- Twenty Thousand Leagues Under The Seas
  MA, USA
```

As you can see, easy as pie! So why not give it a try? (The nice nemo green color is grayscaled for the printed \textit{TUGboat}.)

\section*{2 Coder’s newsletter}

In \textit{Digital Typography} [2], Knuth stresses the importance of stability in \TeX and METAFONT, which ensures identical output across time and across different computer systems. Over the last forty years, this stability has allowed an ecosystem of third-party software to grow around \TeX. To make it easier to develop complex software solutions, we show how coders can integrate Markdown with third-party software in sections 2.1–2.3.

In Section 1, we showed new syntax extensions for Markdown. However, syntax extensions are costly to implement, undermine the minimalism of Markdown, and can never account for all components and concepts a writer may need. Therefore in Section 2.4, we present the concepts of \HTML attributes and attribute contexts, which can be used to define domain-specific dialects of Markdown in \TeX without the need for new syntax extensions.

\subsection*{2.1 Building better APIs with YAML}

In Section 1.4, we showed how authors can include metadata in their Markdown documents using the YAML language. To react to the metadata, we can use a high-level key–value interface in the \texttt{expl3} programming language:\(^7\)

\footnotesize
\begin{verbatim}
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}
---
title: Of *Wolves* & _Grandmas_
author: Little Red Cap
---
When Little Red Cap entered the woods a wolf came up to her.
\end{markdown}
\end{document}
\end{verbatim}
\end{footnotesize}

\footnotesize
\(^4\) github.com/witiko/markdown/issues/91
\(^5\) github.com/witiko/markdown/issues/22
\(^6\) github.com/xvrabcov/md-templates
\(^7\) github.com/witiko/markdown/issues/22

---

\textit{TUGboat}, Volume 43 (2022), No. 1

Markdown 2.15.0: What’s new?
2.2 Passing HTML through to \TeX4ht

Using the \TeX4ht system, we can convert \TeX documents to HTML for publishing on the web. Since \TeX4ht uses \LaTeX for the conversion, it supports the Markdown package out-of-the-box. However, it is still necessary to use correct command-line options depending on which \TeX engine we use. To use Lua\TeX, we can use the \texttt{--lua} option:

$ \texttt{make4ht \ --lua\ document.tex}$

With other \TeX engines, we must use the \texttt{--shell-escape} option, which enables shell access:

$ \texttt{make4ht \ --shell-escape\ document.tex}$

Since version 2.3.0, Markdown has supported the \texttt{html} option, which allows us to use HTML tags in Markdown documents. Since version 2.14.0, Markdown has also supported renderers for HTML tags.

Unless redefined by the user, these renderers will pass any HTML elements through to the output of \TeX4ht, whereas they will be ignored in PDF output:

\begin{document}
\begin{markdown}
Hello \textbf{world}!
\end{markdown}
\end{document}

\begin{document}
\begin{markdown}
Hello <b>world</b>!
\end{markdown}
\end{document}

\begin{document}
\begin{markdown}
Hello \textbf{world}!
\end{markdown}
\end{document}

2.3 Integration with Pandoc

Pandoc is a tool for converting between dozens of document formats. In our proof of concept, we integrate Pandoc with the Markdown package so that we can typeset and style any document format understood by Pandoc directly from \TeX.

To give an example, we have prepared a manual page \texttt{wolf.1} in the roff language:

\texttt{.SH NAME}
\texttt{wolf \- tool for befriending grandmas}
\texttt{.SH SYNOPSIS}
\texttt{.B wolf} \texttt{[\fB-b\fR|\fB--befriend\fR]} \texttt{[\fB-s\fR|\fB--scare\fR]} \texttt{<\fIgrandma\fR>}

Here is how we would typeset our manual page:

\begin{document}
\usepackage[pandoc-to-markdown, emoji]
\markdownSetup{renderers = {\section*{\emoji{wolf}#1}}}
\begin{document}
\begin{pandocInput}[format=man]{wolf.1}
\end{document}
\end{pandocInput}
\end{document}

\begin{document}
\begin{markdown}
Hello \textbf{world}!
\end{markdown}
\end{document}

\begin{document}
\begin{markdown}
Hello \textbf{world}!
\end{markdown}
\end{document}

---

\[\texttt{github.com/witiko/markdown/issues/90}\]

\[\texttt{github.com/drehak/pandoc-to-markdown}\]
NAME
wolf - tool for befriending grandmas

SYNOPSIS
wolf [-b|--befriend] [-s|--scare] <grandma>

Our proof of concept consists of a Lua writer that produces \TeX\ commands corresponding to Pandoc’s abstract syntax tree and a \TeX\ package that maps these commands to the renderers of the Markdown package. A rewrite of our Lua writer in Haskell will be offered as a basis of the upcoming plain \TeX\ writer for Pandoc.\footnote{github.com/jgm/pandoc/issues/1541}

2.4 Actionable attributes and contexts

In Section 1.3, we showed how authors can add HTML attributes to section headings. We can react to the attributes by redefining attribute renderers. Furthermore, the HTML attributes of a Markdown element are surrounded by attribute contexts, which we can use to limit the effects of an attribute:\footnote{github.com/witiko/markdown/issues/91}

\begin{verbatim}
\documentclass{article}
\usepackage[headerAttributes]{markdown}
\markdownSetup{
  renderers = {
    headerAttributeContextBegin = \begingroup,
    headerAttributeContextEnd = \endgroup,
    attributeClassName = {\% \markdownIfSnippetExists{#1}{\% \markdownSetup{snippet=#1}}{}}
  },
}
\markdownSetupSnippet{sans-serif}{
  code = {\% \def\familydefault{\sfdefault}\%
    \fontfamily{\familydefault}\%
    \selectfont
  },
}
\begin{document}
\begin{markdown}
# A section
This section is typeset in a serif typeface.

# Another section {.sans-serif}
This section is typeset in sans-serif ...
\end{markdown}
\end{document}
\end{verbatim}

In Section 3.2, we discuss our plans for other elements of Markdown that may be able to receive HTML attributes in the future.

3 Developer’s newsletter

In the following sections, we describe ideas for improving the Lua parser (3.1 and 3.2), \LaTeX\ interface (3.3 and 3.4), Con\TeXt\ interface (3.5), and Docker images (3.6) of Markdown.

3.1 Smart backslashes and math support

Since Markdown does not detect math at parse time, it can be difficult to write math:

\begin{verbatim}
\documentclass{article}
\usepackage{mathtools}
\usepackage[hybrid]{markdown}
\begin{document}
\begin{markdown}
$$ x_i + y_j = \begin{dcases}
a & b \\
 c & d
\end{dcases} $$
\end{markdown}
\end{document}
\end{verbatim}

Specifically, it is necessary to escape underscores and backslashes, and to be careful with indentation:

\begin{verbatim}
\documentclass{article}
\usepackage{mathtools}
\usepackage[hybrid]{markdown}
\begin{document}
\begin{markdown}
$$ x_i + y_j = \begin{dcases}
 a & b \\
 c & d
\end{dcases} $$
\end{markdown}
\end{document}
\end{verbatim}

Also in our previous article [3, Figure 4], we showed how we can construct a smart lexical preprocessor that only requires the escaping of backslashes.

Markdown 2.15.0: What’s new?
when they precede another escapable character. Furthermore, we can use well-defined heuristics such as dollar signs to detect math at parse time and disable underscores, code listings, and other elements in it.\footnote{\url{github.com/witiko/markdown/issues/61}}\footnote{\url{github.com/witiko/markdown/issues/123}}

\documentclass{article}
\usepackage{mathtools}
\usepackage[smartBackslashes, mathDollars]{markdown}
\begin{document}
\begin{markdown}
\documentclass{article}
\usepackage[linkAttributes]{markdown}
\usepackage{graphicx}
\markdownSetup{
  renderers = {
    linkAttributeContextEnd = \endgroup,
    linkAttributeContextBegin = {\%
    \begingroup \markdownSetup{
      renderers = {
        attributeKeyValue = {\%
          setkeys{Gin}{
            (#1) = (#2),
          }%\%
        },%
      },%
    },%
  },%
}\begin{document}
\begin{markdown}
\markdownRendererHeadingOnePrototype
\markdownRendererHeadingTwoPrototype
\markdownRendererHeadingThreePrototype
\markdownRendererHeadingFourPrototype
\markdownRendererHeadingFivePrototype
\markdownRendererHeadingSixPrototype
\end{document}
\end{markdown}
\end{document}

3.3 Importing \LaTeX\ setup snippets
In our previous article [3, Section 1], we have introduced \LaTeX\ themes and snippets, which can be used to build powerful abstractions in Markdown. Suppose the jdoe/longpackagename/lists \LaTeX\ theme defines the arabic, roman, and alpha setup snippets. If we want to access these snippets by their short names, we must first load the theme and then assign names to the snippets:
\begin{verbatim}
\markdownSetup{
  importSnippets = {
    jdoe/longpackagename/lists = {
      arabic,
      roman,
      alpha as alphabetic,
    },
  },
}
\end{verbatim}

3.4 Advanced renderer definitions in \LaTeX
At the moment, the \markdownSetup \LaTeX\ command only allows the redefinition of one renderer or renderer prototype at a time, which makes it difficult to redefine several renderers or renderer prototypes at once:
\begin{verbatim}
\markdownSetup{
  rendererPrototypes = {
    headingOne = {
      \chapter{#1},
    },
    headingTwo = {
      \section{#1},
    },
    headingThree = {
      \subsection{#1},
    },
    headingFour = {
      \subsubsection{#1},
    },
    headingFive = {
      \paragraph{#1},
    },
    headingSix = {
      \subparagraph{#1},
    },
  },
}
\end{verbatim}

Furthermore, it is difficult to keep some parts of previous definitions without using low-level code:
\begin{verbatim}
\usepackage{etoolbox}
\xpatchcmd
  \markdownRendererHeadingOnePrototype
  \#{1}\#{1}\
\end{verbatim}

\footnote{\url{github.com/witiko/markdown/issues/107}}

Vít Novotný, Dominik Rehák, Michal Hoftich, Tereza Vrabcová
In order to make it easier to redefine renderers and renderer prototypes partially and in bulk, it may be useful to extend the syntax of \markdownSetup.\footnote{\url{github.com/witiko/markdown/issues/121}}
\markdownSetup{
   rendererPrototypes = {
      heading* {#1} = {
         #1
      },
   },
}

### 3.5 Idiomatic \TeX{}X setup

Unlike \LaTeX{}, which has high-level syntax for setting up Markdown, Con\TeX{}X has only a few additions over the plain \TeX{} interface for Markdown. Since version 2.15.0, there has been a concerted effort to extend Markdown, so that it can enumerate and examine its own options, renderers, and renderer prototypes.\footnote{\url{github.com/witiko/markdown/issues/119}} This will make it easier to create and maintain new high-level interfaces for formats other than \LaTeX{}, such as Con\TeX{}X.\footnote{\url{github.com/witiko/markdown/issues/17}}

### 3.6 Additional binary platforms in Docker

Since version 2.10.0, Markdown has been available as the \texttt{witiko/markdown} Docker image.\footnote{\url{hub.docker.com/r/witiko/markdown}} In version 2.15.0, images for \TeX{} Live 2019–2021 are available, which makes it easy to use Markdown for continuous integration with services such as GitHub Actions:

```yaml
name: Typeset a document
on: {push: ~}
jobs:
typeset:
   runs-on: ubuntu-latest
   container:
      image: witiko/markdown:TL2019-historic
   steps:
      - uses: actions/checkout@v2
      - run: latexmk -lualatex document.tex
```

The \texttt{witiko/markdown} Docker image is based on the \texttt{texlive/texlive} Docker image from the Island of \TeX{} [1], which is only available for the \texttt{linux/amd64} platform. This is sufficient for most continuous integration services. However, to allow interactive use of \texttt{witiko/markdown}, it may be useful to add support for multi-platform builds to \texttt{texlive/texlive}.\footnote{\url{gitlab.com/islandoftex/images/texlive/-/issues/15}}

### References


- Vít Novotný
  Studená 453/15
  Brno, 638 00
  Czech Republic
  witiko (at) mail dot muni dot cz
github.com/witiko

- Dominik Rehák
  Legionárska 71
  Trenčín, 911 04
  Slovak Republic
drehak (at) firemail dot cc
github.com/drehak

- Michal Hoftich
  Magdalény Rettigové 4
  Praha, 116 39
  Czech Republic
  michal dot h21 (at) gmail dot com
github.com/michal-h21

- Tereza Vrabcová
  V Aleji 130/30
  Děčín, 405 02
  Czech Republic
  vrabcova dot tereza (at) email dot cz
github.com/xvrabcov
The DuckBoat — Beginners’ Pond: CDs, but not Compact Disks

Herr Professor Paulinho van Duck

Abstract

In this installment, Prof. van Duck will show you some tips & tricks about \texttt{tikz-cd}, a package for high-quality typesetting of commutative diagrams.

1 \TeX.SE has a new moderator!

Hi, (\La)TeX friends!

An exceptional event took place last November: a \TeX.SE moderator’s election, quack!

Moderators are elected for life because they are somewhat compared to Supreme Court judges, who are appointed lifelong (in some countries, at least). Therefore, their elections are quite infrequent.

Of course, the role of moderator is not a life sentence. There is also the possibility of resigning.

Our former moderator Martin Scharrer made that choice last October. Martin had been in that role since \TeX.SE left beta in 2011. He is also the author of many valuable packages, like \texttt{standalone} and \texttt{adjustbox}. We are immensely grateful to him for all his past and future contributions to the \LaTeX community.

I would also like to thank the friends who nominated themselves as candidates in the moderator election. There were a couple of well-known names but even relatively new users. I am particularly glad to have seen people from non-Western cultures, and I hope to have a woman, too, next time.

It was very hard to choose. Eventually, the winner was Werner Grundlingh! (Simply known as Werner on \TeX.SE: \texttt{tex.stackexchange.com/users/5764/werner}).

He needs no introduction for the \TeX.SE users since he is the third in the site ranking, after egreg and David Carlisle. He has always stood out for his patience, good temper, and his craving for helping others. Let me congratulate him. He has the skills to be a great mod, quack!

Unfortunately, there is also bad news. Our dear friend Brent Longborough passed away in December 2021. He gave an outstanding contribution both on \TeX.SE and with his packages and collaborations (for example, \texttt{arara}).

I like to remember his great sense of humor. He defined himself “old-ish IT geezer, young at heart.”

We will miss him!

As for TopAnswers \TeX, \url{https://topanswers.xyz/tex}, the site grows and starts appearing in browsers’ search results. I suggest you try it. In particular, if your question on other sites remains unanswered, you can be sure it will be looked after on TopAnswers \TeX. Moreover, if you have problems with \texttt{beamer}, you will find the great expert samcarter answering there.

I take the occasion to thank her for her help setting the correct fonts in the example of Box 14.

If you are talking with a mathematician about CDs, you almost certainly are not referring to an old-fashioned(?!?) way to listen to music.

The acronym CDs stands for Commutative Diagrams. My math colleagues say they are graphs largely used in category theory. I do not know what category theory is, but I did not ask, in order to avoid a two-hour math lesson, quack!

However, judging by the number of questions about them, they are undoubtedly prevalent.

Box 1 shows a little example. It is the representation of $h \circ f = k \circ g$.

\begin{figure}[h]
\centering
\begin{tikzcd}
A & B \\
C & D
\end{tikzcd}
\end{figure}

Do you find it beautiful? Of course, you do!

In the following, I will show you how to create it with \texttt{tikz-cd}. There are also other packages for drawing commutative diagrams, for instance \texttt{xy}, but in my opinion \texttt{tikz-cd} is the simplest and the most customizable.

2 Quack Guide No. 7

A package for typing beautiful CDs

In a previous article of mine [1], I told you about \texttt{TikZ} matrices. Simply speaking, a \texttt{TikZ} CD is a \texttt{TikZ} matrix with some arrows added.
Let us start examining the code in Box 1.

First of all, you have to load the package. Alternatively, if you have already loaded TikZ in your preamble, you may take advantage of the `cd` library:
\usepackage{tikz}
\usetikzlibrary{cd}

Now you are ready to use the `tikzcd` environment. Please note that I have included it within `$...$` to show it as a displayed equation.

The core of your CD is typed like an ordinary array, with `&` to separate the cells (the vertices of the diagram) and `\` to separate the rows. Pay attention that the last row does not end with `\`; otherwise, you will get an undesired empty line.

The new additions are the commands `\ar`, an abbreviation for `\arrows`. As it is obvious from their name, they allow you to draw your arrows, quack!

The first parameter of `\ar` is the direction: `r`, `l`, `d`, and `u`, which stand for right, left, down, up. You can combine two or more directions. For example, `dr` means your arrow will go diagonally from the current cell, where the `\ar` is positioned, to the one down and right from it. You can also repeat a direction. For instance, `rr` will go from the current cell to the second cell right to it.

What appears between quotes are the labels of the arrows. The option `swap` makes the label be placed on the right side of the arrow, relative to its direction (left is the default).

I will describe how to build and customize your diagram with the available options in the following.

Remember that, in general, options can be set either for a single element; or for the current CD as environment options:
\begin{tikzcd}[\textit{options}]
\end{tikzcd}

or for all the CDs of your document by putting:
\begin{tikzcdset}{\textit{options}}
\end{tikzcdset}

in your preamble.


2.1 About vertices

Since a TikZ CD is actually a TikZ matrix, it inherits all matrix characteristics.

You can set the distance separating columns with `\[width\]` and for rows with `\\[width\]`.

If you would like to change these separators for all the rows/columns of your diagram, you can set them with the options `column sep`/`row sep`.

You may indicate either an explicit dimension or the predefined ones: `tiny`, `small`, `scriptsize`, `normal`, `large`, and `huge`.

---

**Box 2 – Column and row separators**

\begin{tikzcd}
&&B\ar[drr]\&\\
A\ar[urr]\ar[dr]\&\&\&C\\
&\&\&D\ar[rr]\&\&E\ar[ur]
\end{tikzcd}

Please note how, in Box 2, the empty cells are used to position the vertices correctly. But be careful that if a cell is a target point of an arrow, it must have a text. Otherwise, you will get the error: `No shape named ... is known`.

If you want to have an arrow pointing to an empty vertex, use `{}` for one vertex at a time, or the matrix option `nodes in empty cells`, for all the vertices at once.

\begin{tikzcd}
&&X\ar[dl]\ar[dr]\&\&\&Y\ar[ul]\ar[ur]
\end{tikzcd}

If you would like to customize one vertex (for example, to draw it), you can use `\textit{options}` before the node text:

\begin{tikzcd}
A\ar[r]\ar[d]&B\ar[d]
\end{tikzcd}

If you would like to modify all the vertices of your diagram, you can use `\textit{options}`, which is equivalent to `every cell/.append style={\textit{options}}`
whereas

```latex
\begin{tikzcd}
\]
cells={nodes=draw}
\]
A\ar[r]&B
\]
\end{tikzcd}
```

One useful node option is `alias=\{cell name\}`. It allows you to refer to the vertex with \{cell name\} when you are drawing your diagram.

Another way for referring to the vertices is `\{matrix name\}-(\{row number\}-(\{column number\})` where `\{matrix name\}` could be `\tikzcdmatrinxname` or a name you set with `every matrix/.append style=\{name=\{matrix name\}\}`

Within a path you can also use \texttt{\tikztostart} (starting point of the path) and \texttt{\tikztotarget} (target point of the path).

In Section 2.4 I will show how node naming is indispensable for drawing complex diagrams.

### 2.2 Go wild with the arrows

The package `tikz-cd` allows a huge number of arrow tips and shapes.

Box 3 shows some examples. For the complete list, see Section 1.3 of the package documentation.

#### Box 3 – Some arrow types

```latex
\begin{tikzcd}
\]
labels={font=\ttfamily\scriptsize},
column sep=7em, row sep=9ex
\]
A\ar[r, dashleftarrow, \"text\{dashleftarrow\}"
\]
\ar[d, mapsto, \"text\{mapsto\}"
\]
\{rotate=90, anchor=south\}]
B\ar[r, Rightarrow, \"text\{Rightarrow\}"
\]
\ar[d, dash, \"text\{dash\}"
\]
\{rotate=-90, anchor=south\}]
\]
D\ar[r, hookrightarrow, \"text\{hookrightarrow\}" description]
\ar[l, leftarrowtail, \"text\{leftarrowtail\}"
\]
\ar[r, lefttarrowtail] description]
E\ar[r, lefttarrowtail, \"text\{lefttarrowtail\}"
\]
\ar[l, righttarrowup, \"text\{righttarrowup\}"
\]
\ar[u, rightharpoonup, \"text\{rightharpoonup\}"
\]
\end{tikzcd}
```

You can also set options for arrows, at either the environment or document level, with `arrows=\{options\}` which appends options to the default, or `every arrow/.style=\{options\}` which replaces the default.

Let us examine some of them (label options will be explained in the next Section).

If the arrows of `tikz-cd` are not enough for you, you may use all the types provided by Ti\textsc{K}Z, setting `arrow style=tikz`.

For example, if you would like to have stealth arrows throughout your document, you can set

```latex
\usetikzlibrary{arrows.meta}
\tikzcdset{
\]
arrows={>={Stealth}}
\}
```

in your preamble, and all your arrows will appears like this:

When your CD has many vertices, it may be boring or difficult to indicate the correct number of r, l, d, and u you need to reach the target point. The arrow options to/from=\{argument\} may help you. The argument could be in the form \{row number-column number\}; a string of r, l, d, and u; or \{cell name\}.

You may even use to and from together and draw your arrows at the end of the diagram.

The following code is an alternative way to produce the diagram in Box 2:

```latex
\begin{tikzcd}
\]
column sep=3pt,
row sep=scriptsize
\]
&\ar[r, dashleftarrow, \"text\{dashleftarrow\}"
\]
\ar[d, mapsto, \"text\{mapsto\}"
\]
\{rotate=90, anchor=south\}]
\]
&\ar[r, Rightarrow, \"text\{Rightarrow\}"
\]
\ar[d, dash, \"text\{dash\}"
\]
\{rotate=-90, anchor=south\}]
\]
\]
&\ar[r, hookrightarrow, \"text\{hookrightarrow\}" description]
\ar[l, leftarrowtail, \"text\{leftarrowtail\}"
\]
\ar[r, lefttarrowtail] description]
E\ar[r, lefttarrowtail, \"text\{lefttarrowtail\}"
\]
\ar[l, righttarrowup, \"text\{righttarrowup\}"
\]
\ar[u, rightharpoonup, \"text\{rightharpoonup\}"
\]
\end{tikzcd}
```
It may happen that you need to shift your arrows to better position them. You can easily do this with \texttt{shift left/right=(dimension)} (the parameter is optional).

It is also possible to use \texttt{xshift=(dimension)}, \texttt{yshift=(dimension)} or \texttt{shift=(coordinate)}.

Box 4 shows some shiftings. The dotted lines represent the default positions. Some arrows are drawn in gray and the CD is enlarged with \texttt{\scalebox} to better visualize the positioning. For the \texttt{ampersand replacement} option see Section 2.4, Box 14.

Box 4 – Shifting

You can also explicitly set a starting/ending point with \texttt{start/end anchor=
\{coordinate transformations\}(anchor)}

Box 5 – Start/end anchor

As you see in Box 5, if you are tired of straight arrows, you can use options inherited from TikZ to bend them:

- \texttt{bend right/left=(angle)}, to curve the arrow
- \texttt{out/in=(angle)}, to set the angle at which the arrow leaves/reaches the vertices
- \texttt{loop}, possibly with \texttt{above/right/below/left} or setting the in/out angles
- \texttt{looseness=(number)} to choose the “level of bending”.

For more details, see Sections 74.3 and 74.4 of the TikZ manual [4].

Box 6 shows some of these options. Please note the environment option \texttt{bezier bounding box}. It is useful to set the correct bounding box when there is looseness in order to avoid unwanted empty space around your diagram. To use it just add the \texttt{bbox} TikZ library [2] in your preamble:

\begin{tikzcd}
\scalebox{1.5}{% 
\begin{tikzcd}
\& Y \& \ar[dr, shift left, gray] \ar[from=dr, shift left]
\ar[dr, dash, dotted, thick] \ar[dl, xshift=1.5ex, gray] \ar[from=dl, yshift=1.5ex]
\ar[dl, dash, dotted, thick]\&
X \& \ar[rr, shift={(2pt,-4pt)}] \ar[rr, dash, dotted, thick] \&
\& Z
\end{tikzcd}%
\end{tikzcd}
\end{verbatim}

Box 6 – Bent arrows

Another useful option is \texttt{shorten=(dimension)}. It allows to shorten or, if \texttt{(dimension)} is negative, elongate your arrow.

This option acts on both sides of the arrows, if you would like to shorten/elongate only one side, use the standard TikZ options \texttt{shorten >=(dimension)} or \texttt{shorten <=(dimension)}.
2.3 Adding labels

The general syntax for labels is
\[
\{\text{label text}\}\{\text{label options}\}
\]
where the curly braces are mandatory only if \text{label text} or \text{label options} contain a comma.

You can set the label options at the environment level specifying
\texttt{labels=(options)}
which appends the style to the default, or
\texttt{every label/.style=(options)}
which replaces the default style.

When including them in \texttt{\tikzcdset} in your preamble, they will be valid at the document level, as usual.

In Box 3, for example, I set the label font at the environment level. Note that since the labels are typed in math mode by default, to have them in text mode you have to use \texttt{\text{label text}}.

The options
\texttt{rotate=(degree)} and \texttt{anchor=(anchor)}
are necessary to put the label along the vertical ar-
The former may be achieved, for example, taking advantage of the \TeX{}Z possibility to set the matrix’s column style.

For the latter, the appropriate coordinates can be used in a `from/to` or the path can be designed with `to path`.

**Box 10 – Column style**

\begin{tikzcd}
\begin{align*}
\text{a+b} & \ar[r] & \text{c} \\
\downarrow & & \downarrow \\
\text{d} & \ar[r] & \text{e-f}
\end{align*}
\end{tikzcd}

Here, as usual in \TeX{}Z, `-|` means “horizontal coordinate of the point before - and vertical coordinate of the point after |”. For further details see Section 3.2 of the package documentation [3].

If the standard alignment is OK for you, but you would like, for example, to have the same length of the horizontal arrows without calculating it manually, you can set the node width to a given dimension.

In the following, the width of the longest node text is used, taking advantage of the option `text width=width(`string`)`.

**Box 11 – Node width**

\begin{tikzcd}
\begin{align*}
\text{a+b} & \ar[r] & \text{c} \\
\downarrow & & \downarrow \\
\text{d} & \ar[r] & \text{e-f}
\end{align*}
\end{tikzcd}

For complex diagrams, it is possible to add code after your diagram is drawn, with `remember picture` and `overlay` or `execute at end picture`.

The difference between these two methods is that what you draw with the former is not included in the bounding box of the picture. To visualize this, compare the alignment of the following two examples.

In Box 12, the entire picture is centered, including what gets drawn at end (the “PvD” and curves).

**Box 12 – Execute at end picture**

\begin{tikzcd}
\begin{align*}
\text{a+b} & \ar[r] & \text{c} \\
\downarrow & & \downarrow \\
\text{d} & \ar[r] & \text{e-f}
\end{align*}
\end{tikzcd}

In Box 13, only the actual CD is centered, what is added afterwards is not.

**Box 13 – Remember picture and overlay**

\begin{tikzcd}
\begin{align*}
\text{a+b} & \ar[r] & \text{c} \\
\downarrow & & \downarrow \\
\text{d} & \ar[r] & \text{e-f}
\end{align*}
\end{tikzcd}

The DuckBoat — Beginners’ Pond: CDs, but not Compact Disks
Choose one or the other method depending on your needs.

Last but not least, when you use a `tikzcd` in a `beamer` frame or as argument to a command, you get the error:

Single ampersand used with wrong catcode.

You could also have strange errors or undesired alignments when your vertices or labels are math matrices or text tables or, in general, any environment that uses `&`.

These problems can be easily solved using `ampersand replacement=\&` and then using `<macro name>` instead of `&`.

### Box 14 – Ampersand replacement

```latex
\begin{document}
\begin{frame}
\begin{tikzcd}
X \ar[r]\ar[d,"\begin{tabular}{cc}
a & b \\
c & d 
\end{tabular}"]\& \begin{pmatrix}
\alpha & \beta \\
\gamma & \delta 
\end{pmatrix}\ar[d]
\end{tikzcd}
\end{frame}
\end{document}
```

```
\begin{tikzcd}
X \ar[r]\ar[d,"\begin{tabular}{cc}
a & b \\
c & d 
\end{tabular}"]\& \begin{pmatrix}
\alpha & \beta \\
\gamma & \delta 
\end{pmatrix}\ar[d]
\end{tikzcd}
```

3 Conclusions

If you have been affected by Stendhal syndrome after looking at the commutative diagrams drawn with `tikz-cd`, remember:

```
For a quack math, ask a TikZ duck!
```

References


PvD

\[ A \ar[r] \ar[d] & B \ar[d] \\
C \ar[r] & D \]

\begin{tikzpicture}
\node[left=of \tikzcdmatrixname] (leftofm) {PvD};
\draw[->] plot [smooth, tension=7] coordinates
{(\tikzcdmatrixname-1-1.160) (leftofm.north east)}
{(\tikzcdmatrixname-1-1.200)};
\draw[->] plot [smooth, tension=7] coordinates
{(mycd-2-1.160) (leftofm.south east)}
{(mycd-2-1.200)};
\end{tikzpicture}

Herr Professor Paulinho van Duck
Making open source textbooks, and diagrams with AlDraTex

Seth D. Bergmann

Abstract
This article describes a new paradigm for the creation of textbooks, using \LaTeX. Macros for the automatic generation of figures and diagrams are described.

1 Introduction
Textbooks have traditionally been produced by publishing companies which provide many services, and revenue, to authors. These services include:

- Editors responsible for the acquisition and production process
- Copyeditors responsible for formatting, proofreading the text for errors and making stylistic improvements
- Graphic artists responsible for the production of technical figures and diagrams
- Sales representatives responsible for the marketing of the book

All these services, plus other corporate employees, bricks and mortar, etc., contribute to the extremely high cost of textbooks for readers.

This article proposes a new paradigm for the creation of textbooks, based on the open source model of software development [2]. Open source software is developed by many developers who may be at remote locations, cooperating on the Internet. In addition to distributing the software product, the developers also distribute the source documents, making it easy for others to make corrections, enhancements, or extensions to the original product. Since there is no cost for the product, there is no revenue for the developers. Some common examples of open source software: Java, Apache, and \LaTeX.

Developers who contribute to a well-known successful product are rewarded by the status associated with the product, and can perhaps leverage this status with job offers or promotions.

In applying the open source paradigm to textbooks, authors can collaborate on the generation of high-quality books. As the books become widely adopted, errata are found and corrected, and other authors are able to add sections and chapters to improve the book. The authors (usually teachers or professors, but also students) receive no direct compensation, but if the book is widely adopted, they receive recognition for their contributions, which can be a positive factor in applying for jobs, tenure, promotion, etc.

The most significant impediment to open-source textbooks is the production of high-quality figures and diagrams. This article makes the claim that diagrams can be drawn with \LaTeX macros, rather than traditional drawing tools. The distinction is that I am proposing the use of a markup language as opposed to WYSIWYG tools.

The development of the graphics macros can be a difficult, and time-consuming process. However, once the macros are perfected, they can be used to draw many diagrams with a professional appearance. If a minor stylistic change is needed, (e.g., arrowheads in state diagrams need to be enlarged) all that is needed is a change to the macro that draws state diagrams, and all such diagrams are regenerated automatically.

When several authors are collaborating on a textbook, it is not necessary that they all have the expertise needed to develop new macros. This could be the responsibility of one developer, with expertise in the graphics commands, to produce new macros as needed by the other authors.

This paper lists four currently available open source textbooks (and a fifth in development) and shows some examples of the diagrams that are generated automatically.

2 Some currently available open source textbooks
I have developed four open source textbooks, which are currently being distributed by the Campbell Library at Rowan University, with a total of over 40,000 downloads worldwide. They are available at rdw.rowan.edu/oer:

<table>
<thead>
<tr>
<th>Title</th>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler Design:</td>
<td>25,843</td>
</tr>
<tr>
<td>Theory, Tools, and Examples [3]</td>
<td></td>
</tr>
<tr>
<td>Computer Organization with MIPS [4]</td>
<td>11,739</td>
</tr>
<tr>
<td>Introduction to Computer Science with Java [6]</td>
<td>4,757</td>
</tr>
<tr>
<td>Computer Science Principles (Java, Python, and C++ editions) [5]</td>
<td>1,398</td>
</tr>
<tr>
<td>Cryptology with Bitcoin and Blockchain [in development]</td>
<td>0</td>
</tr>
</tbody>
</table>

(Number of downloads as of March 2022.)
I’d like to single out Allen Downey, at Olin College of Engineering, who has also published several open source textbooks [7]. He has been one of the first, if not the first, to promote this new paradigm, making educational resources affordable for students and school districts everywhere.

Another developer of open source textbooks is Jim Hefferon of Saint Michael’s College in Vermont, who has authored books on the Theory of Computation [10] and Linear Algebra [9]. Hefferon has made extensive use of \( \LaTeX \) graphics packages to draw figures and diagrams.

3 Diagrams

In this section I expose some of the more complex diagrams that can be drawn with special-purpose macros. I use the graphics packages DraTex and ADraTex, developed at Ohio State University by Eitan Gurari [8] in the 1990s; they are available in \( \TeX \) Live and on CTAN (ctan.org/pkg/dratex). These packages include fairly primitive structures such as circles, lines, rectangles, and text boxes. They also include state diagrams, trees, grids, and other structures, with parameters controlling the size and appearance.\(^1\)

As with \( \LaTeX \), the DraTex and ADraTex packages are extensible: new macro commands can be defined using existing macro commands. I have used this feature in developing macros for the examples shown in this section.

3.1 Charts, trees, and other diagrams with ADraTex

The ADraTex macros for chart and tree diagrams support diagrams with various attributes and formats, by specifying parameter values in the macro call. ADraTex also allows the user to define other diagrams.

3.2 Chart diagrams

ADraTex has macros which draw charts:

**Bar charts** allow variable size charts, with various properties on the bars, including 3-dimensional bars and tailored or painted bars, and labels on the bars and axes.

3.3 Tree diagrams

Trees, in various formats, may be drawn with ADraTex. The user simply specifies the label on each node and the number of children. Nodes may be in various shapes, including circle, oval, rectangle, and text-only. The format of the tree may allow for horizontal–vertical edges or straight edges, and optional labels on the edges. Trees may be oriented vertically or horizontally.

3.4 Other diagrams

As an extensible language, with ADraTex it is possible to define nodes of any shape for the diagrams described above, as well as macros for drawing state diagrams, such as those needed for finite automata.

3.5 Logic diagrams

The *Computer Organization* textbook [4] makes extensive use of diagrams. The primitive drawing commands of DraTex can be used to produce diagrams of logic gates—AND, OR, XOR, NOT, etc.

These gates can then be connected with each other to form higher level components such as encoders, decoders, multiplexers, adders, and arithmetic/logic units. An example of a logic diagram is shown in Figure 1.

One commonly used logic diagram is a canonical sum-of-products diagram, corresponding to a logic expression which is the logical OR of several logical ANDs. I developed a macro to draw sum-of-products diagrams with up to four variables. An example is shown in Figure 3. The user of this macro need only specify the names of the variables and the boolean true/false values for each product in the sum-of-products expression to be diagrammed.

3.6 Karnaugh maps

Another important diagram for the *Computer Organization* textbook is the Karnaugh Map (or K-Map). It is used to reduce a logic expression to its simplest sum-of-products form. The macro which draws K-Maps needs input for the positions of 1’s (or don’t-cares) in the map, as well as the grouping of the 1’s. In theory a macro could deduce the groupings, but this feature is not currently available. The macro can draw K-Maps with three or four variables. An example of a K-Map with four variables is shown in Figure 5.

---

\(^1\) Other graphics packages with similar capabilities include \texttt{tikz}, available on github, and \texttt{metapost}, available from the Tex User Group.
3.7 Object diagrams
The most interesting diagram macro that I have developed is for the object diagram construct commonly used in textbooks on object-oriented programming. I used this macro in the textbook *Introduction to Computer Science with Java* [6].

In an object-oriented language, such as Java or C++, the state of an object is determined by the instance variables\(^2\) in the object’s class. Each instance variable in an object may store either primitive data or a reference\(^3\) to another object. This is a recursive definition, and thus the object diagram, which is a visual representation of an object, is a recursive diagram. Consequently the macro which draws object diagrams is also recursive. An example of an object diagram is shown in Figure 7.

4 Examples of diagrams
Examples of diagrams which were described above are shown on the following pages. In each case the commands which generated the diagram are shown in the next figure. The full macro definitions and packages are available at cs.rowan.edu/~bergmann/books.

5 Summary
In this article I have presented a new paradigm for the production of free open-source textbooks. This paradigm has been used to produce four computer science textbooks, with a fifth book in development. The main contribution of this work is the capability of drawing figures and diagrams with \LaTeX{} macros. These macros are based on the extensible graphics packages DraTex and AlDraTex.

References

\[ xy + yz \]

Figure 1: An implementation of the boolean function \(xy + yz\) using logic gates

\DiagramSpec({\Inp & \Gate & \Wire})
\Diagram
%%% Inputs: Var, xLoc, yLoc
(x,-50,30 & y,-50,0 & z,-50,-30)

%%% Gates: ID, type, inputs, xLoc, yLoc
(And1,0,2,0,30,, & And2,0,2,0,-30,, & Or,1,2,50,0,0,)

%%% Wires: srcID, targetID, input#
(x, And1,1 & y, And1,2 & y, And2,1 & z,And2,2 & And1,Or,1 & And2,Or,2)
\MoveToNode(Orout,1,0.5) \Move(50,0) \FcNode(result) \Edge(Orout,result) \EdgeLabel(--$xy + yz$--)

Figure 2: \LaTeX{} code used to draw Figure 1


\[ \text{\copyright Seth D. Bergmann} \\
\text{Rowan University} \\
\text{Glassboro, NJ, USA} \\
\text{bergmann (at) rowan dot edu} \\
\text{https://cs.rowan.edu/~bergmann/} \]

Making open source textbooks, and diagrams with AlDraTex
Figure 3: A sum of products logic diagram for the expression $xy'zw' + y'x + wx'$

\DiagramSpec{\SOP & \Ins & \Ands}
\Diagram
(4,3,xy'zw'+y'x+wx') \ll 4 \text{ variables, 3 terms}
(x & y & z & w) \ll \text{Variable names}
\text{Terms: #vars, var, negated}
(4,x,1,y,0,z,1,w,0 \ll \text{xy}^{'},z)w'
2,y,0,x,1,0,0,0,0 \ll \text{y}'x
2,w,1,x,0,0,0,0,0 \ll \text{wx}'
)

Figure 4: \LaTeX{} code used to draw Figure 3

\begin{array}{cccc}
  yz & yz & yz & yz \\
  00 & 01 & 11 & 10 \\
\end{array}

\begin{array}{|c|c|c|c|}
\hline
wx=00 & 0 & 1 & 0 & 0 \\
\hline
wx=01 & 0 & 0 & 1 & 1 \\
\hline
wx=11 & 0 & 0 & 1 & 1 \\
\hline
wx=10 & 1 & 0 & 0 & 1 \\
\hline
\end{array}

Figure 5: A K-map for the boolean expression $w'x'y'z + w'xyz + w'xyz' + wx'yz' + wx'y'z' + wx'yz$. A 1x2 group and a 2x2 group are identified. The minimized expression is $xy + wx'z' + w'x'y'z$.

\KmapFourTop(0,1,0,0, 0,0,1,1, 32)
\KmapFourBot(0,0,1,1, 1,0,0,1, 32)
\Group(4,7,32,) \ll \text{xy}
\Group(h,10,32,h) \ll \text{wx}'z'$

Figure 6: \LaTeX{} code used to draw Figure 5

Seth D. Bergmann
**Figure 7:** An object diagram showing the value of the variable `myMap` storing a reference to a map, after three entries have been added.

```latex
\Objective (5, myMap, 
  HashMap~$<\text{String}, \text{Student}>$, 
  3, keyPrims, keyObjs, 
  valuePrims, valueObjs) 
```

**Figure 8:** \LaTeX{} code used to draw Figure 7.

Making open source textbooks, and diagrams with AlDraTex
Automatically removing widows and orphans with lua-widow-control

Max Chernoff

Abstract

The lua-widow-control package, for plain LuaTeX/LuaLaTeX/ConTeXt/OpTeX, removes widows and orphans without any user intervention. Using the power of LuaTeX, it does so without stretching any glue or shortening any pages or columns. Instead, lua-widow-control automatically lengthens a paragraph on a page or column where a widow or orphan would otherwise occur.

To use lua-widow-control, all that most users need do is place \usepackage{lua-widow-control} in their preamble. No further changes are required.

1 Motivation

\TeX provides top-notch typesetting: even 40 years after its first release, no other program produces higher quality mathematical typesetting, and its paragraph-breaking algorithm is still state-of-the-art. However, its page breaking is not quite as sophisticated as its paragraph breaking and thus suffers from some minor issues.

Unmodified \TeX has only two familiar ways of dealing with widows and orphans: it can either shorten a page by one line, or it can stretch vertical whitespace. \TeX was designed for mathematical and scientific typesetting, where a typical page has multiple section headings, tables, figures, and equations. For this style of document, \TeX’s default behaviour works quite well, since the slight stretching of whitespace between the various document elements is nearly imperceptible; however, for prose or other documents composed almost entirely of paragraphs, there is little vertical whitespace to stretch.

Since no ready-made and fully-automated solution to remove widows and orphans from all types of documents was available, I decided to create lua-widow-control.

2 What are widows and orphans?

2.1 Widows

A “widow” occurs when the majority of a paragraph is on one page or column, but the last line is on the following page or column. It not only looks quite odd for a lone line to be at the start of the page, but it makes a paragraph harder to read since the separation of a paragraph and its last line disconnects the two, causing the reader to lose context for the widowed line.

2.2 Orphans

An “orphan” occurs when the first line of a paragraph is at the end of the page or column preceding the remainder of the paragraph. They are not as distracting for the reader, but they are still not ideal. Visually, widows and orphans are about equally disruptive; however, orphans tend not to decrease the legibility of a text as much as widows, so many authors choose to ignore them.

See Figure 1 for a visual reference.

2.3 Broken hyphens

“Broken” hyphens occur whenever a page break occurs in a hyphenated word. These are not related to widows and orphans; however, breaking a word across two pages is at least as disruptive for the reader as widows and orphans. \TeX identifies broken hyphens in the same ways as widows and orphans, so lua-widow-control treats broken hyphens in the same way.

3 History and etymology

The concept of widows and orphans is nearly as old as printing itself. In [13], a printers manual from 1683, we have:

Nor do good Compoſiters account it good Workmanship to begin a Page with a Break-line, unleſs it be a very ſhort Break, and cannot be gotten in the foregoing Page; but if it be a long Break, he will let it be the Direction-line of the fore-going Page, and Set his Direc-tion at the end of it. (p. 226)

3.1 Widows

However, the terms “widow” and “orphan” are much newer. The earliest published source that I could find referencing “widows” in typography is Webster’s New International Dictionary from 1934. However, no one — not even the editors of the dictionary [3] —
seems to know how it got there. Even then, the
definition is somewhat different than it is now:
widow, n. c. Print. A short line or single word
carried over from the foot of one column or
page to the head of a succeeding column or
page. [3]
Contrast this with the modern definition:
Typography. A short line of text (usually
one consisting of one word or part of a word)
which falls undesirably at the end of a para-
graph, esp. one set at the top of a page or
column. [16]
which includes a single lone line of any length.

3.2 Orphans
The term “orphan” is even more confusing. Its ini-
tial usage seems to have occurred some time after
“widow” [3], and it is given many contradictory de-
definitions. Most sources define an orphan as a first
line at the bottom of the page and a widow as the
last line at the top [2, 3, 4, 6, 9, 12, 14, 16]; how-
ever, some sources define these two terms as exact opposites of each other, with a widow as a first line
at the bottom of the page and an orphan as the
last line! [1, 3, 5, 14, 18] This usage is plain wrong;
nevertheless, it is sufficiently common that you need
to be careful when you see the terms “widow” and
“orphan”.

3.3 Clubs
The \TeXbook never refers to “orphans” as such;
rather, it refers to them as “clubs”. This term is
remarkably rare: I could only find a single source
published before The TeXbook — a compilation arti-
cle about the definition of “widow” — that mentions
a “club line”:

The Dictionary staff informs me that they have no example of the use of the word widow
in the typographical sense. […]
Mr. Watson of the technical staff says that the Edinburgh printing houses referred to it
as a “clubline”. [3, p. 4]
To my knowledge, a ‘widow’, or ‘widow-line,’
is a short line, forming the end of a paragraph,
which is carried over from the foot of a page or
column to the top of the succeeding one. […]
To my personal knowledge, in typographical parlance in Edinburgh, Scotland, the ‘widow’
is called a ‘club-line.’ [3, p. 23]
Both quotes above are from separate authors,
and they each define a “club” like we define “widow”,
not an “orphan”. In addition, they both mention that
the term is only used in Scotland. Even the extensive
OED — which lists 17 full definitions and 103 subde-
definitions for the noun “club” — doesn’t recognize the
phrase. [15]
I spent a few hours searching through Google
Books and my university library catalogue, but I
could not find a single additional source. If anyone
has any more information on the definition of a “club
line” or why Knuth chose to use this archaic Scottish
term in \TeX, please let me know!

4 Pagination in \TeX
Let’s move on to looking at how \TeX treats these
widows and orphans.

4.1 Algorithm
It is tricky to understand how lua-widow-control
works if you aren’t familiar with how \TeX breaks
pages and columns. For a full description, you should
consult Chapter 15 of The \TeXbook [9] (“How \TeX
Makes Lines into Pages”); however, this goes into
much more detail than most users require, so here is
a very simplified summary of \TeX’s page breaking
algorithm:
\TeX fills the page with lines and other objects
until the next object will no longer fit. Once no more
objects will fit, \TeX will align the bottom of the last
line with the bottom of the page by stretching any
available vertical spaces if (in LATEX) \texttt{\flushbottom}
is set; otherwise, it will break the page and leave the
bottom empty.

However, some objects have penalties attached.
Penalties encourage or discourage page breaks from
occurring at specific places. For example, LATEX
sets a negative penalty before section headings to
encourage a page break there; conversely, it sets a
positive penalty after section headings to discourage
breaking.

To reduce widows and orphans, \TeX sets weakly-
positive penalties between the first and second lines
of a paragraph to prevent orphans, and between the
penultimate and final lines to prevent widows.

One important note: once \TeX begins breaking
a page, it never goes back to modify any content
on the page. Page breaking is a localized algorithm,
without any backtracking.

4.2 Behaviour
Merely describing the algorithm doesn’t allow us to
intuitively understand how \TeX deals with widows
and orphans.

Due to the penalties attached to widows and
orphans, \TeX tries to avoid them. Widows and
orphans with small penalties attached — like \TeX’s

Automatically removing widows and orphans with lua-widow-control
default values of 150 — are only lightly coupled to the rest of the paragraph, while widows and orphans with large penalties — values of 10 000 or more — are treated as infinitely bad and are thus unbreakable. Intermediate values behave just as you would expect, discouraging page breaks proportional to their value.

When \TeX goes to break a page, it tries to avoid breaking at a location with a high penalty. How it does so depends on a few settings:

4.2.1 \texttt{\flushbottom} and \texttt{\normalbottom}

With the settings \texttt{\normalbottom} (Plain \TeX) or \texttt{\flushbottom} (\LaTeX), \TeX is willing to stretch any glue on the page by an amount roughly commensurate to the magnitude of the penalty: for small \texttt{\clubpenalty} and \texttt{\widowpenalty} values, \TeX will only slightly stretch the glue on the page before creating a widow or orphan; for very large penalties, \TeX will stretch the glue by a near-infinite amount.

This corresponds to the “Stretch” column in Figure 2. It is the default behaviour of Plain \TeX, and of the standard \LaTeX classes when the \texttt{twocolumn} option is given.

4.2.2 \texttt{\raggedbottom}

When \texttt{\raggedbottom} is set, \TeX won’t stretch any glue. Instead, for sufficiently-high \texttt{\clubpenalty} and \texttt{\widowpenalty} values, \TeX will shorten the page or column by one line in order to prevent the widow or orphan from being created.

This corresponds to the “Shorten” column in Figure 2 and is the default behaviour of the \LaTeX classes when the \texttt{twocolumn} option is not given.

5 \texttt{\looseness}

Before we can continue further, we need to discuss one more \TeX command: \texttt{\looseness}. The following is excerpted from Chapter 14 of [9] (“How \TeX Breaks Paragraphs into Lines”):

If you set \texttt{\looseness=1}, \TeX will try to make the current paragraph one line longer than its optimum length, provided that there is a way to choose such breakpoints without exceeding the tolerance you have specified for the badness of individual lines. Similarly, if you set \texttt{\looseness=2}, \TeX will try to make the paragraph two lines longer; and \texttt{\looseness=-1} causes an attempt to make it shorter. […]

For example, you can set \texttt{\looseness=1} if you want to avoid a lonely “club line” or “widow line” on some page that does not have sufficiently flexible glue, or if you want the total number of lines in some two-column document to come out to be an even number.

It’s usually best to choose a paragraph that is already pretty “full”, i.e., one whose last line doesn’t have much white space, since such paragraphs can generally be loosened without much harm. You might also want to insert a tie between the last two words of that paragraph, so that the loosened version will not end with only one “widow word” on the orphans line; this tie will cover your tracks, so that people will not find it hard to detect the fact that you have tampered with the spacing. On the other hand, \TeX can take almost any sufficiently long paragraph and stretch it a bit, without substantial harm.

The widow and orphan removal strategy suggested in the second paragraph works quite well; however, it requires manual editing each and every time a page or paragraph is rewritten or repositioned.

6 Alternate removal strategies

There have been a few previous attempts to improve upon \TeX’s previously-discussed widow and orphan-handling abilities; however, none of these have been able to automatically remove widows and orphans without stretching any glue or shortening any pages.

The articles “Strategies against widows” by Paul Isambert [6] and “Managing forlorn paragraph lines” by Frank Mittelbach [11] both begin with comprehensive discussions of the methods of preventing widows and orphans. They agree that widows and orphans are bad and ought to be avoided; however, they differ in their solutions. Strategies proposes an output routine that reduces the length of facing pages by one line when necessary to remove widows and orphans, while Managing proposes that the author manually rewrites or adjusts \texttt{\looseness} when needed.

The post “Paragraph callback . . .” by jeremie [7] contains a file \texttt{widow-assist.lua} that automatically detects which paragraphs can be safely shortened or lengthened by one line. Mittelbach’s \texttt{widows-and-orphans} package [12] alerts the author to the pages that contain widows or orphans. Combined, these packages make it simple for the author to quickly remove widows and orphans by adjusting the values of \texttt{\looseness}; however, it still requires the author to make manual source changes after each revision.

Another article by Mittelbach [10] suggests an fully-automated solution to remove widows and orphans. This would seem to offer a complete solution; however, it requires multiple passes, an external tool, and has not yet been publicly released.

Max Chernoff
lua-widow-control can remove most widows and orphans from a document, without stretching any glue or shortening any pages.

It does so by automatically lengthening a paragraph on a page where a widow or orphan would otherwise occur. While \TeX breaks paragraphs into their natural length, lua-widow-control is breaking the paragraph 1 line longer than its natural length. \TeX's paragraph is output to the page, but lua-widow-control's paragraph is just stored for later. When a widow or orphan occurs, lua-widow-control can take over. It selects the previously-saved paragraph with the least badness; then, it replaces \TeX's paragraph with its saved paragraph. This increases the text block height of the page by 1 line.

Now, the last line of the current page can be pushed to the top of the next page. This removes the widow or the orphan without creating any additional work.

\parskip=0pt
\clubpenalty=0
\widowpenalty=0

lua-widow-control can remove most widows and orphans from a document, without stretching any glue or shortening any pages.

It does so by automatically lengthening a paragraph on a page where a widow or orphan would otherwise occur. While \TeX breaks paragraphs into their natural length, lua-widow-control is breaking the paragraph 1 line longer than its natural length. \TeX's paragraph is output to the page, but lua-widow-control's paragraph is just stored for later. When a widow or orphan occurs, lua-widow-control can take over. It selects the previously-saved paragraph with the least badness; then, it replaces \TeX's paragraph with its saved paragraph. This increases the text block height of the page by 1 line.

Now, the last line of the current page can be pushed to the top of the next page. This removes the widow or the orphan without creating any additional work.

\parskip=0pt plus 1fill
\clubpenalty=10000
\widowpenalty=10000

Figure 2: A visual comparison of various automated widow-handling techniques.
lua-widow-control is essentially a combination of widow-assist.lua [7] and widows-and-orphans [12] (although its implementation is independent of both): when the `outputpenalty` value indicates that a widow or orphan has occurred, Lua is used to find a stretchable paragraph. What lua-widow-control mainly adds on top of these packages is automation: it eliminates the requirement for any manual adjustments or changes to your document’s source.

7 Visual comparison

Although TeX’s page breaking algorithm is reasonably straightforward, it can lead to complex behaviour when widows and orphans are involved. The usual choices, when rewriting is not possible, are to ignore them, stretch some glue, or shorten the page. Figure 2 has a visual comparison of these options, which we’ll discuss in the following:

7.1 “Ignore”

As you can see, the last line of the page is on a separate page from the rest of its paragraph, creating a widow. This is usually highly distracting for the reader, so it is best avoided for the reasons previously discussed.

7.2 “Shorten”

This page did not leave any widows, but it did shorten the previous page by one line. Sometimes this is acceptable, but usually it looks bad because pages will then have different text-block heights. This can make the pages look quite uneven, especially when typesetting with columns or in a book with facing pages.

7.3 “Stretch”

This page also has no widows and it has a flush bottom margin. However, the space between each pair of paragraphs had to be stretched.

If this page had many equations, headings, and other elements with natural space between them, the stretched out space would be much less noticeable. TeX was designed for mathematical typesetting, so it makes sense that this is its default behaviour. However, in a page with mostly text, these paragraph gaps look unsightly.

Also, this method is incompatible with grid typesetting, where all glue stretching must be quantised to the height of a line.

7.4 “lua-widow-control”

lua-widow-control has none of these issues: it eliminates the widows in a document while keeping a flush bottom margin and constant paragraph spacing.

To do so, lua-widow-control lengthened the second paragraph by one line. If you look closely, you can see that this stretched the interword spaces. This stretching is noticeable when typesetting in a narrow text block, but is mostly imperceptible with larger widths.

lua-widow-control automatically finds the “best” paragraph to stretch, so the increase in interword spaces should almost always be minimal.

8 Installation and standard usage

The lua-widow-control package was first released in October 2021. It is available in the default installations of both MiKTeX and TeX Live, although you will need recent versions of either.

You may also download lua-widow-control manually from either CTAN,¹ the ConTeXt Garden,² or GitHub,³ although it is best if you can install it through your TeX distribution.

As its name may suggest, lua-widow-control requires LuaTeX⁴ regardless of the format used. With that in mind, using lua-widow-control is quite simple:

Plain TeX

\input lua-widow-control

OpTeX

\load[lua-widow-control]

\lwcsetup

\setuplwc [lua-widow-control]

ConTeXt

\setupmodule[lua-widow-control]

And that’s usually enough. Most users won’t need to do anything else since lua-widow-control comes preconfigured and ready-to-go.

9 Options

Nevertheless, lua-widow-control does have a few options.

lua-widow-control tries very hard to have a “natural” user interface with each format, so how you set an option heavily depends on how you are running lua-widow-control. Also note that not every option is available in every format.

Some general guidelines:

Plain TeX/OpTeX Some options are set by modifying a register, while others must be set manually using \directlua.

\lwcsetup

\lwcsetup[\lwcsetup]

ConTeXt Always use \setuplwc.

¹ ctan.org/pkg/lua-widow-control

² modules.contextgarden.net/cgi-bin/module.cgi?action=view/id=127

³ github.com/gucci-on-fleek/lua-widow-control/releases/latest/

⁴ Or LuaMetaTeX in the case of ConTeXt.

Max Chernoff
9.1 Disabling
You may want to disable `lua-widow-control` for certain portions of your document. You can do so with the following commands:

Plain TeX/OpTeX
\lwcdisable

I\TeX
\lwcsetup{disable}

Con\TeXt
\setuplwc[state=stop]

This prevents `lua-widow-control` from stretching any paragraphs that follow. If a page has earlier paragraphs where `lua-widow-control` was still enabled and a widow or orphan is detected, `lua-widow-control` will still attempt to remove the widow or orphan.

9.2 Enabling
`lua-widow-control` is enabled as soon as the package is loaded. If you have previously disabled it, you will need to re-enable it to save new paragraphs.

Plain TeX/OpTeX
\lwcanable

I\TeX
\lwcsetup{enable}

Con\TeXt
\setuplwc[state=start]

9.3 Automatically disabling
You may want to disable `lua-widow-control` for certain commands where stretching is undesirable such as section headings. Of course, manually disabling and then enabling `lua-widow-control` multiple times throughout a document would quickly become tedious, so `lua-widow-control` provides some options to do this automatically for you.

`lua-widow-control` automatically patches the default \texttt\lwcsetup{disable} \texttt\lwcsetup{enable} \texttt\setuplwc[state=start] \texttt\setuplwc[state=stop]. You don't need to touch these. Any others, though, you'll need to patch yourself.

Plain TeX/OpTeX\lwcdisablecmd{\macro}

I\TeX\lwcsetup{disablecmds={
\csname one\endcsname, \csname two\endcsname}}

Con\TeXt\prependtoks\lwc@patch@pre
\to\everybefore\hook
\prependtoks\lwc@patch@post
\to\everyafter\hook

9.4 \emergencystretch
`lua-widow-control` defaults to an \emergencystretch value of 3 em for stretched paragraphs, but you can configure this.

`lua-widow-control` will only use the \emergencystretch when it cannot lengthen a paragraph in any other way, so it is fairly safe to set this to a large value. TeX accumulates badness when \emergencystretch is used [8], so it's pretty rare that a paragraph that requires any \emergencystretch will actually be used on the page.

Plain TeX/OpTeX \lwcemergencystretch={\dimension}

I\TeX\lwcsetup{emergencystretch={\dimension}}

Con\TeXt\setuplwc[emergencystretch={\dimension}]

9.5 Penalties
You can also manually adjust the penalties that \TeX assigns to widows and orphans. Usually, the defaults are fine, but there are a few circumstances where you may want to change them.

Plain TeX/OpTeX \widowpenalty={\integer}
\clubpenalty={\integer}
\brokenpenalty={\integer}

I\TeX\lwcsetup{widowpenalty={\integer}}
\lwcsetup{orphanpenalty={\integer}}
\lwcsetup{brokenpenalty={\integer}}

Con\TeXt\setuplwc[widowpenalty={\integer}]
\setuplwc[orphanpenalty={\integer}]
\setuplwc[brokenpenalty={\integer}]

The value of these penalties determines how much \TeX should attempt to stretch glue before passing the widow or orphan to `lua-widow-control`. If you set the values to 1 (default), \TeX will stretch nothing and immediately trigger `lua-widow-control`; if you set the values to 10 000, \TeX will stretch infinitely and `lua-widow-control` will never be triggered. If you set the value to some intermediate number, \TeX will first attempt to stretch some glue to remove the widow or orphan; only if it fails will `lua-widow-control` come in and lengthen a paragraph. As a special case, if you set the values to 0, both \TeX and `lua-widow-control` will completely ignore the widow or orphan.

9.6 \nobreak behaviour
When `lua-widow-control` encounters an orphan, it removes it by moving the orphaned line to the next page. The majority of the time, this is an appropriate solution. However, if the orphan is immediately preceded by a section heading (or \nobreak/\penalty 10 000), `lua-widow-control` would naively separate a section heading from the paragraph that follows. This is almost always undesirable, so `lua-widow-control` provides some options to configure this.

Plain TeX/OpTeX \directlua{\lwc.nobreak\_behaviour="\value"}

I\TeX\lwcsetup{nobreak={\value}}

Con\TeXt\setup\lwc[nobreak={\value}]

Automatically removing widows and orphans with `lua-widow-control`
The default value, `keep`, keeps the section heading with the orphan by moving both to the next page. The advantage to this option is that it removes the orphan and retains any `nobreaks`; the disadvantage is that moving the section heading can create a large blank space at the end of the page.

The value `split` splits up the section heading and the orphan by moving the orphan to the next page while leaving the heading behind. This is usually a bad idea, but exists for the sake of flexibility.

The value `warn` causes `lua-widow-control` to give up on the page and do nothing, leaving an orphaned line. `lua-widow-control` warns the user so that they can manually remove the orphan.

See figure 3 for a visual reference.

### 9.7 Maximum cost

`lua-widow-control` ranks each paragraph on the page by how much it would “cost” to lengthen that paragraph. By default, `lua-widow-control` selects the paragraph on the page with the lowest cost; however, you can configure it to only select paragraphs below a selected cost.

If there aren’t any paragraphs below the set threshold, then `lua-widow-control` won’t remove the widow or orphan and will instead issue a warning.

Plain \TeX\/\LaTeX \texttt{\lwcmaxcost\{integer\}}

\LaTeX \texttt{\lwcsetup\{max-cost\{integer\}\}}

\ConTeXt \texttt{\setuplwc\{maxcost\{integer\}\}}

Based on my testing, `max-cost` values less than 1000 cause completely imperceptible changes in interword spacing; values less than 5000 are only noticeable if you are specifically trying to pick out the expanded paragraph on the page; values less than 15000 are typically acceptable; and larger values may become distracting. `lua-widow-control` defaults to an infinite `max-cost`, although the “strict” and “balanced” modes set the values to 5000 and 10000 respectively.

### 10 Presets

As you can see, `lua-widow-control` provides quite a few options. Luckily, there are a few presets that you can use to set multiple options at once. These presets are a good starting point for most documents, and you can always manually override individual options.

Currently, these presets are \LaTeX-only.

\LaTeX \texttt{\lwcsetup\{\textit{preset}\}}

#### 10.1 default

If you use `lua-widow-control` without any options, it defaults to this preset. In default mode, `lua-widow-control` takes all possible measures to remove widows and orphans and will not attempt to stretch any vertical glue. This usually removes >95% of all possible widows and orphans. The catch here is that this mode is quite aggressive, so it often leaves behind some fairly “spacey” paragraphs.

This mode is good if you want to remove (nearly) all widows and orphans from your document, without fine-tuning the results.

#### 10.2 strict

`lua-widow-control` also offers a strict mode. This greatly restricts `lua-widow-control`’s tolerance and makes it so that it will only lengthen paragraphs where the change will be imperceptible.

The caveat with strict mode is that — depending on the document — `lua-widow-control` will be able to remove less than a third of the widows and orphans. For the widows and orphans that can’t be automatically removed, a warning will be printed to your terminal and log file so that a human can manually fix the situation.

This mode is good if you want the best possible typesetting and are willing to do some manual editing.

#### 10.3 balanced

Balanced mode sits somewhere between default mode and strict mode. This mode first lets \TeX{} stretch a little glue to remove the widow or orphan; only if that fails will it then trigger `lua-widow-control`. Even then, the maximum paragraph cost is capped. Here, `lua-widow-control` can usually remove 90% of a document’s potential widows and orphans, and it does so while making a minimal visual impact.

This mode is recommended for most users who care about their document’s typography. This mode is not the default since it doesn’t remove all widows and orphans: it still requires a little manual intervention.

### 11 Compatibility

The `lua-widow-control` implementation is almost entirely in Lua, with only a minimal \TeX{} footprint. It doesn’t modify the output routine, inserts/floats,
Table 1: lua-widow-control options set by each mode.

<table>
<thead>
<tr>
<th>Option</th>
<th>default</th>
<th>balanced</th>
<th>strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-cost</td>
<td>∞</td>
<td>10000</td>
<td>5000</td>
</tr>
<tr>
<td>emergencystretch</td>
<td>3em</td>
<td>1em</td>
<td>0pt</td>
</tr>
<tr>
<td>nobreak</td>
<td>keep</td>
<td>keep</td>
<td>warn</td>
</tr>
<tr>
<td>widowpenalty</td>
<td>1</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>orphanpenalty</td>
<td>1</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>brokenpenalty</td>
<td>1</td>
<td>500</td>
<td>1</td>
</tr>
</tbody>
</table>

\everypar, and it doesn’t insert any whatsits. This means that it should be compatible with nearly any \TeX package, class, and format. Most changes that lua-widow-control makes are not observable on the \TeX side.

However, on the Lua side, lua-widow-control modifies much of a page’s internal structure. This should not affect any \TeX code; however, it may surprise Lua code that modifies or depends on the page’s low-level structure. This does not matter for Plain \TeX or \LaTeX, where even most Lua-based packages don’t depend on the node list structure; nevertheless, there are a few issues with Con\TeXt.

Simple Con\TeXt documents tend to be fine, but many advanced Con\TeXt features rely heavily on Lua and can thus be disturbed by lua-widow-control. This is not a huge issue — the lua-widow-control manual is written in Con\TeXt — but lua-widow-control is inevitably more reliable with Plain \TeX and \LaTeX than with Con\TeXt.

Finally, keep in mind that adding lua-widow-control to a document will almost certainly change its page break locations.

11.1 Formats

lua-widow-control runs on all known Lua\TeX-based formats: Plain Lua\TeX, Lua\LaTeX, Con\TeXt MkIV, Con\TeXt MkXL/LMTX, and Op\TeX. Unless otherwise documented, all features should work equally well in all formats.

11.2 Columns

Since \TeX and the formats implement column breaking and page breaking through the same internal mechanisms, lua-widow-control removes widows and orphans between columns just as it does with widows and orphans between pages.

lua-widow-control is known to work with the \LaTeX class option \texttt{twocolumn} and the two-column output routine from Chapter 23 of [9].

11.3 Performance

lua-widow-control runs entirely in a single pass, without depending on any .aux files or the like. Thus, it shouldn’t meaningfully increase compile times. Although lua-widow-control internally breaks each paragraph twice, modern computers break paragraphs near-instantaneously, so you are not likely to notice any slowdown.

11.4 \v TeX penalties

Knuth’s original \TeX has three basic line penalties: \texttt{\interlinepenalty}, which is inserted between all lines; \texttt{\clubpenalty}, which is inserted after the first line; and \texttt{\widowpenalty}, which is inserted before the last line. The \v TeX extensions [20] generalize these commands with a syntax similar to \texttt{\parshape}: with \texttt{\widowpenalties} you can set the penalty between the last, second last, and \texttt{n}th last lines of a paragraph; \texttt{\interlinepenalties} and \texttt{\clubpenalties} behave similarly.

lua-widow-control makes no explicit attempts to support these new -penalties commands. Specifically, If you give a line a penalty that matches either \texttt{\widowpenalty} or \texttt{\clubpenalty}, lua-widow-control will treat the lines exactly as it would a widow or orphan. So while these commands won’t break lua-widow-control, they are likely to lead to some unexpected behaviour.

12 Short last lines

When lengthening a paragraph with \texttt{\looseness}, it is common advice to insert ties (\texttt{\-}) between the last few words of the paragraph to avoid overly-short last lines [9]. lua-widow-control does this automatically, but instead of using ties or \texttt{\hboxes}, it uses the \texttt{\parfillskip} parameter [9, 21]. When lengthening a paragraph (and only when lengthening a paragraph — remember, lua-widow-control doesn’t interfere with \TeX’s output unless it detects a widow or orphan), lua-widow-control sets \texttt{\parfillskip} to 0pt plus 0.8\texttt{hsize}. This normally makes the last line of a paragraph be at least 20% of the overall paragraph’s width, thus preventing ultra-short lines.

13 How it works

lua-widow-control uses a fairly simple algorithm to eliminate widows and orphans, but there are a few subtleties.

13.1 Setup

lua-widow-control sets the parameters \texttt{\clubpenalty}, \texttt{\widowpenalty}, and \texttt{\brokenpenalty} to sentinel values of 1. This will signal to lua-widow-control when...
a widow or orphan occurs, yet it is small enough that it won’t stretch any glue.

lua-widow-control also enables Lua\TeX{}’s microtypographic extensions [19]. This isn’t strictly necessary; however, it significantly increases the number of paragraphs that can be acceptably “loosened”.

That is all that happens on the \TeX{} end. The rest of lua-widow-control is pure Lua.

### 13.2 Paragraph breaking

First, lua-widow-control hooks into the paragraph breaking process, before any output routines or page breaking.

Before a paragraph is broken by \TeX{}, lua-widow-control grabs the unbroken paragraph. Then lua-widow-control breaks the paragraph one line longer than its natural length and stores it for later. It does this in the background, without interfering with how \TeX{} breaks paragraphs into their natural length.

After \TeX{} has broken its paragraph into its natural length, lua-widow-control appears again. Before the broken paragraph is added to the main vertical list, lua-widow-control “tags” the first and last nodes of the paragraph using a Lua\TeX{} attribute. These attributes associate the previously-saved lengthened paragraph with the naturally-typedet paragraph on the page.

### 13.3 Page breaking

lua-widow-control intercepts $\backslash$box255 (the $\backslash$box output by \TeX{}) immediately before the output routine runs, after all the paragraphs have been typeset.

First, lua-widow-control looks at the \texttt{outputpenalty} of the page or column. If the page was broken at a widow or orphan, the \texttt{outputpenalty} will be equal to either the \texttt{widowpenalty} or the \texttt{clubpenalty}. If the \texttt{outputpenalty} does not indicate a widow or orphan, lua-widow-control will stop and return $\backslash$box255 unmodified to the output routine, and \TeX{} continues as normal.

Otherwise, we assume that we have a widow or orphan on the page, meaning that we should lengthen the page by 1 line. We iterate through the list of saved paragraphs to find the lengthened paragraph with the least cost. After we’ve selected a good paragraph, we traverse through the page to find the original version of this paragraph — the one that unmodified \TeX{} originally typeset. Having found the original paragraph, we splice in the lengthened paragraph in place of the original.

Since the page is now 1 line longer than it was before, we pull the last line off the page to bring it back to its original length, and place that line onto the top of \TeX{}’s “recent contributions” list. When the next page begins, this line will be inserted before all other paragraphs, right at the top. Now, we can return the new, widow-free page (updated \texttt{box255}) to the output routine, which proceeds as normal.

### 14 Choosing the “best” paragraph

As we discussed previously, lua-widow-control lengthens the paragraph with the lowest cost. However, assigning a cost to each paragraph is not quite as simple as it sounds. Before we look at how lua-widow-control assigns costs, let’s look at how \TeX{} scores paragraphs when breaking them naturally.

#### 14.1 How \TeX{} scores paragraphs

All glue in \TeX{} has a certain natural size: the size that it would be in an ideal scenario. However, most glue also has stretch and shrink components so that the glue can change in size to adapt to its surroundings. For each line, \TeX{} individually sums the total stretch/shrink for the line and the stretch/shrink that was actually used. We define the stretch/shrink ratio $r$ as the quotient of the stretch/shrink used and the stretch/shrink available. Then the badness $b$ of a line is approximately defined as

$$b = 100r^3.$$  

This is the badness referenced in the commonly-seen Underfull \texttt{\hbox} (badness 1234) warnings that \TeX{} produces.

\TeX{} calculates the badness for each line individually; however, we also need to assess the paragraph as a whole. To do so, \TeX{} defines the demerits for a whole paragraph $d$ as approximately\footnote{We ignore any additional demerits or penalties that \TeX{} may add.} the sum of the squared badnesses for each line. The natural paragraph that \TeX{} breaks is the one that minimizes $d$.

One important thing to realize is that demerits grow incredibly fast: demerits are proportional to the sixth power of glue stretch. This means that you can expect to see extremely large demerit values, even for a relatively “good” paragraph.

#### 14.2 Possible cost functions

Now, let’s return to how lua-widow-control assigns costs to each paragraph. This is surprisingly more complicated than it sounds, so we’ll go through a few possible cost functions first.

Here, we use $c$ for the cost of a paragraph, $d$ for the total demerits, and $l$ for the number of lines (\texttt{prevgraf}).
14.2.1 The original implementation
The original implementation of lua-widow-control used the very simple cost function
\[ c = d. \]
This cost function works reasonably well, but has one major issue: it doesn’t take into account the number of lines in the paragraph. The demerits for a paragraph is the sum of the demerits for each line. This means this cost function will prefer using shorter paragraphs since they tend to have fewer demerits. However, long paragraphs tend to have much more available glue stretch, so this strategy can lead to suboptimal solutions.

14.2.2 Scaling by the number of lines
Once I realized this issue, I tried correcting it by dividing by the number of lines in the paragraph to get the average demerits instead of the total demerits:
\[ c = \frac{d}{l}. \]
This works better than the previous function, but still has an issue. If we have a fairly bad ten-line paragraph with total demerits \(10d\) and an almost-equal length two-line paragraph with total demerits \(2d + 1\), then by this cost function, the ten-line paragraph will have a lower cost and will be chosen. This means that our page now has ten bad lines instead of two bad lines, which is not ideal.

14.2.3 Current implementation
Our first cost function, \(c = dl^0\), doesn’t consider the number of lines at all, while our second cost function, \(c = dl^{-1}\), considers the number of lines too much. Splitting the difference between the two functions, we get the current implementation:
\[ c = \frac{d}{\sqrt{l}}. \]
I didn’t arrive at this function through any sort of scientific testing; rather, I picked the simplest function that I could think of that satisfies the following two properties:
- Given a long paragraph and a short paragraph with different average badnesses per line, prefer the one with the least average badness.
- Given two paragraphs with equal average badnesses per line, prefer the shorter one.

15 Quantitative analysis
Let’s look at some statistics for lua-widow-control. For testing, I downloaded the top ten books on Project Gutenberg \(^6\) converted them to LATEX using pandoc, concatenated them into a single article file, and compiled twice. This gives us a PDF with 1381 pages, 15692 paragraphs, 61865 lines, and 399 widows and orphans (if they aren’t removed).

This is a fairly challenging test: almost every third page has a widow or orphan, over half of the paragraphs have two lines or fewer, and the text block is set to the fairly wide article defaults. An average document is much less challenging for lua-widow-control, so we can consider this to be a worst-case scenario.

15.1 Widows and orphans removed
When we run LATEX with its default settings on the file, 179 (47%) of the widows and orphans are removed. When we add lua-widow-control with default settings, we remove 392 (98%). Switching to strict mode, we can only remove 52 (13%) of the widows and orphans. In balanced mode, we remove 348 (87%). See figure 5 for a visual comparison.

15.2 Paragraph costs
The last section showed us that lua-widow-control is quite effective at removing widows and orphans, so now let’s look at the paragraphs that lua-widow-control expands. As \TeX{} processes a document, lua-widow-control is recording the costs for the naturally-broken and expanded versions of each paragraph in the document. Costs don’t mean that much on their own, but a lower cost is always better.

As you can see in figure 6, the lengthened paragraphs tend to have much higher costs than the naturally-broken paragraphs. This is not surprising, since (as we’ve seen) a paragraph’s demerits scale with the sixth power of glue stretch, so even a small amount of glue stretch can cause a huge increase in demerits.

The empty space on the left of the “long” line is from the paragraphs that Lua\TeX{} was unable to lengthen at any cost. Lua\TeX{} assigns these paragraphs zero demerits, so they disappear on a logarithmic plot.

15.3 Lengthening vs. shortening paragraphs

Figure 7 shows the number of paragraphs that Lua\widow-control could potentially stretch or shrink. The one-line paragraphs are broken out separately since this test sample has an anomalous number of them. Otherwise, we can see that Lua\widow-control is capable of stretching the majority of paragraphs.

We can also see that of non-single-line paragraphs, only about 8% of paragraphs can only be shrunk (the last segment of figure 7), and this is in a document where 13% of paragraphs have at least eight lines. Most documents rarely have such long paragraphs, and it is these long paragraphs that are the easiest to shrink.

Because of this, Lua\widow-control doesn’t even attempt to shrink paragraphs; it only stretches them.

16 Known issues

Lua\widow-control is quite stable these days, a few issues remain:

- When a three-line paragraph is at the end of a page forming a widow, Lua\widow-control will remove the widow; however, it will leave an orphan. This issue is inherent to any process that removes widows through paragraph expansion and is thus unavoidable. Orphans are considered to be better than widows [2], so this is still an improvement.

- Sometimes a widow or orphan cannot be eliminated because no paragraph has enough stretch. Sometimes this can be remediated by increasing Lua\widow-control’s \emergencystretch; however, some pages just don’t have any suitable paragraph.

- Long paragraphs with short words tend to be stretchier than short paragraphs with long words since these long paragraphs have more interword glue. Narrow columns also stretch more easily than wide columns since you need to expand a paragraph by less to make a new line.

- When running under LuaMeta\TeX{} (Con\TeX{}), the log may contain many lines like “\texttt{latex warning > tex: left parfill skip is gone”.

Max Chernoff
These messages are completely harmless (although admittedly quite annoying).

- \TeX{} may warn about overfull \vbox{}es on pages where lua-widow-control removed a widow or orphan. This happens due to the way that lua-widow-control corrects for the $\texttt{\textbackslash prevdepth}$ when replacing paragraphs. It does not actually produce an overfull \vbox{}, but there is a warning nevertheless. You can set $\texttt{\textbackslash vfuzz=2\textbackslash pt}$ to hide the warning.

- lua-widow-control only attempts to expand paragraphs on a page with a widow or orphan. A global system like in [10] would solve this; however, this is both NP-complete [17] and impossible to solve in a single pass. Very rarely would such a system remove widows or orphans that lua-widow-control cannot.

17 Conclusion

All this probably makes lua-widow-control look quite complicated, and this is true to some extent. However, this complexity is hidden from the end user: as stated at the outset, most users merely need to place $\texttt{\usepackage{lua-widow-control}}$ in their \LaTeX{} document preamble, and lua-widow-control will remove all the troublesome widows and orphans, without needing any manual intervention.

Should you have any issues, questions, or suggestions for lua-widow-control, please visit the project’s GitHub page: github.com/gucci-on-fleek/lua-widow-control. Any feedback is greatly appreciated!

References

[20] The \N\U\G\O\B\T\S Team. The \vTeX{} manual, Feb. 1998. ctan.org/pkg/etex

○ Max Cherno昀昀
Calgary, Alberta
Canada
mseven at telus dot net

Automatically removing widows and orphans with lua-widow-control
**l3build: The beginner’s guide**

Joseph Wright

1 Introduction

For package authors, creating a release is a regular process, ideal for automation. There are several steps to creating a release to CTAN, for example ensuring documentation is updated, structuring an archive correctly and actually uploading the material.

Some time ago, the LaTeX Team extended their existing basic scripts to create an independent tool, l3build, which can cover all of those tasks. Most significantly, it included features to run comprehensive tests: this aspect was previously covered for *TUGboat* (2014, **35**-3, pp. 287–293). Here, I will give a more general overview of the tool, looking at how it can help package authors create releases in a quick and reliable manner.

2 l3build at the command line

With a modern TeX system, l3build is available as a command at the command line/terminal. It understands ⟨targets⟩, ⟨options⟩ and ⟨arguments⟩.

l3build ⟨target⟩ [⟨options⟩] [⟨arguments⟩]

The ⟨target⟩ is the task we want l3build to carry out. The most common ones are:

- check Runs one or more automated tests
- save Saves the result of one or more tests
- doc Typesets documentation
- ctn Creates a zip file ready to send to CTAN
- upload Sends a zip file to CTAN
- install Installs the package in the local texmf tree (there is also uninstall to reverse this)

The particular ⟨options⟩ which apply depend on the ⟨target⟩. For example, when running the check target, l3build will normally finish all of the tests then report the results. However, particularly when used with an automated system, one might want the tests to halt as soon as there is an error. That is available using the --halt-on-error option, which is also available as the one-letter version -H.

Some targets require one or more ⟨arguments⟩. For example, to save test results, you have to give the name of the test(s). Some targets take an optional ⟨argument⟩: doc is a good example, as you can limit this to a specific PDF (where your project has several PDFs, this can be useful). Finally, some targets do not need arguments at all: install is an example.

3 Configuration: the build.lua file

The configuration of l3build for a project is controlled by a file called build.lua, which should be present in the main directory. This is a Lua file, and so *can* contain sophisticated programming. However, for a large number of use cases, the requirements are simply to set either string variables or tables of strings. That means that for many projects, the build.lua file will comprise just a few short lines, and requires no insight into Lua programming.

Only one line is absolutely required: one to tell l3build the name of the package. This is specified as the module string:

```
module = "mypkg"
```

By the way, Lua will allow us to mark strings using either single or double quotes. I favour double ones, and only use single quotes if the string itself contains a double quote, but it’s purely personal preference.

The standard settings in l3build are based around using one or more .dtx files extracted using an .ins file. They also assume that the documentation is in the .dtx files. One common structure with larger packages is to separate out the documentation from the code, so to have a .tex file to typeset. This can be covered using

```
typesetfiles = {"*.tex"}
```

or if we want to specify only specific files, for example:

```
typesetfiles = 
    {"mypkg-doc.tex",
     "mypkg-example-a.tex"
    }
```

Here, we are using a Lua table: these can hold a variety of data, but all we need to know here is that we can use a comma-separated list of names inside braces.

If the project we are working on doesn’t use the .dtx format, we need to tell l3build the name(s) of our source files, and that it can skip unpacking:

```
sourcefiles = {"*.def", "*.sty"}
unpackfiles = {} 
```

Or we might unpack some files that are not on the standard list, in which case we need to tell l3build to install them:

```
installfiles = {"*.def", "*.sty"}
```

The standard settings for l3build assume that all of the source files are in the same directory as the build.lua file. Some authors prefer a more complex structure. For example, for LaTeX itself there are lots of documentation files, so they are inside a subdirectory:

```
docfiledir = "/doc"
```

You can do the same with your source files, for example if you want your main directory to hold just build.lua (and probably a README.md):
The system can cope with more complex layouts, for example with subdirectories. One new feature that can help with these more tricky cases is `tdsdirs`, which lets l3build simply copy an entire directory ‘as is’. We tell the system the name of the directory, and where it matches up with in the TeX installation tree. For example, if we wanted to use the above `source` directory in its entirety, and install it into the `tex` tree, we would use

```latex
\texttt{tdsdirs = \{source = "tex\}}
```

In this case, all of the files are used.

We will see later that there are settings that apply to tests, to creating CTAN releases, and for more advanced functions.

### 4 Setting up simple tests

The core mechanism for creating tests in l3build uses the fact that documents can write to the `.log` and extract information to verify that our code has worked. That can broadly be done in two ways: deliberately writing information to the `.log`, or using `\showoutput` or similar to place the result of some typesetting operation into the file.

What is also needed is a way to mark those parts of the `.log` that are of interest, and to normalise system-dependent information, such as paths, to make the results as portable as possible. Some of this is carried out by l3build itself, with the macro parts of the process implemented in the source file `regression-test.tex`. All the commands provided by the latter have all-uppercase names, to minimise the chance of clashes with normal commands.

For the case where it is possible to save a result in a macro, counter or similar, the easiest approach to testing is to write these using `\TYPEOUT`.

```latex
\input{regression-test}
\documentclass{article}
\usepackage{mypkg} % The package to test
\begin{document}
\START
\TEST{A first test}{}% The test to check
  \mypkfunctionA{input-tokens} %
  \outputmacro
  \TYPEOUT{\outputmacro}
}\TEST{A second test}{}% 
  \mypkfunctionB
  \{input-tokens\} %
  \{more-input-tokens\} %
  \outputmacro
  \TYPEOUT{\outputmacro}
}\END
```

Nothing before `\START` will be recorded, which makes it a good way to skip the preamble. We can skip small parts of the input using the pair `\OMIT` and `\TIMO`. The run here is stopped using `\END` as we are not interested in the typesetting of pages: this basically kills the TeX run and saves a bit of time.

The alternative approach is to look at TeX’s output tracing, either using a box or `\showoutput`.

```latex
\input{regression-test}
\documentclass{article}
\usepackage{mypkg} % The package to test
\showoutput
\begin{document}
\START
\% Assume the commands produce typeset output
\mypkfunctionA{input-tokens}
\mypkfunctionB
\{input-tokens\} %
\{more-input-tokens\}
\newpage
\OMIT
\end{document}
```

Here, we can use `\OMIT` to skip over the information at the end of a TeX run: here we have used `\end{document}` as this allows the `\*` and `.aux` files, etc., to be created. If you are relying on information passed using this mechanism, you might need to set

```
checkruns = 2
```

or some higher value.

The input files for tests, `.lvt` files, should be saved inside a directory `testfiles` within the project directory. The test results are then saved using

```
l3build save \{names\}
```

where the `{names}` are the file names of the test inputs, but with the extension omitted.

With the standard settings, tests are run using `pdflatex`, `xelatex` and `luatex`, and using the LaTeX format. Using formats other than LaTeX is outside of the scope of this short guide, but running with multiple engines is a common requirement. To save an engine-specific test result, we use the `--engine` (or `-e`) option

```
l3build save -e\{engine1\},\{engine2\} \{names\}
```

This will be needed most commonly when testing typeset output: there are fundamental differences between the three common engines. When running

```
l3build check
```

the system will use engine-specific results if they exist, and otherwise will assume that they all follow the ‘standard’ engine: this is normally `pdflatex`.

---

l3build: The beginner’s guide
If you would rather just use one engine for tests, you can set

\texttt{checkengines = \{"pdftex"\}}

in your \texttt{build.lua} file. For Unicode-only work, in contrast, you might want

\texttt{checkengines = \{"xetex", "luatex"\}}

where the first entry given will then be the ‘standard’ engine.

5 Customising typesetting

There is only one command used for typesetting documentation: it can be set using the \texttt{typesetexe} setting. This is typically set to \texttt{pdflatex}: notice that this is a typesetting \textit{command} not an \textit{engine}.

As for tests, the number of typesetting runs can be set, using the \texttt{typesetruns} setting. More complex adjustment of the typesetting run is possible: \texttt{l3build} provides a set of basic operation functions (such as ‘run Biber’), and these can be combined to make defined workflows. This aspect requires some Lua programming and is therefore beyond the scope of this short guide.

6 Building CTAN releases

The standard settings will collect up all sources and typeset files, plus any \texttt{README.md}, and create a zip file to send to CTAN. You can also pack a \texttt{TDS}-ready zip: this feature is activated using the setting

\texttt{packtdszip = true}

Uploading to CTAN requires some settings to ‘fill out the form’ for administration. As an example, \texttt{l3build} itself has the configuration shown in Figure 1. The \texttt{[[...]]} syntax creates a multi-line string in Lua.

The information in \texttt{uploadconfig} is used by the \texttt{upload} target, which needs two key pieces of information: an email address and a release string. This will be requested by \texttt{l3build} if not given at the command line

\texttt{l3build upload --email \{email\} \{tag\}}

You can check that your upload is valid, without actually sending it, by using the \texttt{--dry-run} option on the command line. (This option also works for the \texttt{install} target.)

7 Advanced features

Using a mixture of Lua programming and additional variables, a wide range of effects can be achieved. These include

- Supporting plain \TeX{} and Con\TeX{}t testing
- Automatically updating version strings and copyright in sources using the \texttt{tag} target
- Using multiple setups to run tests for different aspects of functionality
- Placing installed files in different parts of the \TeX{} tree
- Testing the PDFs produced by typesetting

Of these, the ability to automatically tag files is probably of the broadest interest. However, as sources files are extremely varied, this does require some Lua programming; that takes us beyond the scope of this short article. For details of this and the other more advanced features, please consult the \texttt{l3build} manual.

8 Example \texttt{build.lua} files

8.1 A basic project: one \texttt{.dtx} and one \texttt{.ins}

The most basic setup, following the model used by the \LaTeX{} Team, is to have your code and documentation in a single \texttt{.dtx} file, which has a matching \texttt{.ins} file and (probably) a \texttt{README.md}, all in the same directory. For this, the \texttt{build.lua} file can be a single line:

\texttt{module = "mypkg"}

That’s it: \texttt{l3build} will handle everything else based on its standard settings.

8.2 A ‘self-extracting’ \texttt{.dtx} file

Some people like to combine their \texttt{.ins} file into their \texttt{.dtx}; that is easy to support.

\texttt{module = "mypkg"}

\texttt{unpackfiles \{"*.dtx"\}}

8.3 Documentation separate from sources

With larger projects, you may want your documentation in one or more \texttt{.tex} files separate from the code. Assuming you also want to typeset your code, you’d go with

\texttt{module = "mypkg"}

\texttt{typesetfiles \{"*.dtx", "*.tex"\}}

8.4 Not using DocStrip, and non-standard file types

Not everyone wants to use DocStrip, and while it won’t hurt to leave unpacking enabled, we might well want to skip it. At the same time, we might have some non-standard file types: here some \texttt{.def} files and one \texttt{.lua} file.

\texttt{module = "mypkg"}

\texttt{installfiles = \{"*.def", "mypkg.lua", "*.sty"\}}

\texttt{unpackfiles = \{\}}

---

1 I don’t recommend this structure. You are unlikely to need to send your source by email to someone, and the only real benefit of a single-source approach is for that type of ‘classical’ distribution.
uploadconfig = {
    author = "The LaTeX Team",
    license = "lppl1.3c",
    summary = "A testing and building system for (La)TeX",
    topic = {"macro-supp", "package-devel"},
    ctanPath = "/macros/latex/contrib/l3build",
    repository = "https://github.com/latex3/l3build/",
    bugtracker = "https://github.com/latex3/l3build/issues",
    update = true,
    description = [[
        The build system supports testing and building (La)TeX code, on Linux, macOS, and Windows systems. The package offers:
        * A unit testing system for (La)TeX code;
        * A system for typesetting package documentation; and
        * An automated process for creating CTAN releases.
    ]]
}

Figure 1: uploadconfig for l3build itself

8.5 Source files in different directories

Some developers like to have their sources in different directories inside their project. This likely goes with having separate files for typesetting.

module = "mypkg"
docfiledir = "doc"
sourcefiledir = "source"
typesetfiles = {"*.tex"}

9 Summary of key settings

There are a large number of more specialised settings available in l3build. Table 1 summarises some of the most commonly-used ones. There is a full list in the package documentation.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td>Name of the package</td>
</tr>
<tr>
<td>installfiles</td>
<td>List of files to place in the texmf tree</td>
</tr>
<tr>
<td>sourcefiles</td>
<td>List of sources/pre-extracted files</td>
</tr>
<tr>
<td>typesetfiles</td>
<td>List of sources to typeset</td>
</tr>
<tr>
<td>unpackfiles</td>
<td>List of .ins files to DocStrip</td>
</tr>
<tr>
<td>docfiledir</td>
<td>Location of typeset sources</td>
</tr>
<tr>
<td>sourcefiledir</td>
<td>Location of code sources</td>
</tr>
<tr>
<td>tdsdirs</td>
<td>Table of locations to install directly</td>
</tr>
<tr>
<td>checkengines</td>
<td>List of engines for test runs</td>
</tr>
<tr>
<td>checkruns</td>
<td>Number of (L)TeX runs for testing</td>
</tr>
<tr>
<td>typesetexe</td>
<td>Program to typeset documentation</td>
</tr>
<tr>
<td>typesetruns</td>
<td>Number of (B)TeX runs for typesetting</td>
</tr>
<tr>
<td>packtdsdir</td>
<td>Switch to build TDS-style zip file</td>
</tr>
<tr>
<td>uploadconfig</td>
<td>Table of information for uploading</td>
</tr>
</tbody>
</table>

Table 1: Summary of key settings

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

l3build: The beginner’s guide
Abstract
Most articles that describe how to use the \texttt{glossaries} package consider a single sorted list or possibly multiple lists where each list has a different set of entries (terms, symbols, abbreviations, etc.). However, some documents may instead have each term described in the main matter, with references to the term linking back to that point in the document rather than to a summary list. Alternatively (or additionally) a document may have multiple lists consisting of the same set of entries ordered in different ways.

The examples here were compiled with \texttt{glossaries v4.49 [9]}, \texttt{mfirstuc v2.07 [10]}, \texttt{glossaries-extra v1.47 [8]} and \texttt{bib2gls v2.8 [6]}. Some features and commands are not available in earlier versions. Also some earlier versions have bugs causing unexpected results.

1 The Book of Poisons

\textit{The Book of Poisons} by Stevens and Bannon [3] is an excellent guide for crime fiction writers. It also provides a good example of a mixture of standalone entries (where each entry, in this case the toxic substance, is described in the main matter rather than in a summary list) and repeated lists in the back matter with just the name and no description (ordered by method of administration, by form, by symptoms, by toxicity and by time taken to react). It also has the more traditional lists (a bibliography and a simple glossary of medical terms) which aren’t under discussion here. Finally, there’s an index, which could be implemented using the standard \texttt{index} and associated commands, but since it’s possible to create the index as a by-product of using \texttt{bibgls} for the standalone entries, this will also be covered here.

The book consists of numbered chapters for each particular type of poison (household chemicals, plants, animals and so on). Each chapter is divided into unnumbered (sub) sections describing each poison, using a consistent structure listing:

\begin{itemize}
\item \textbf{Scientific Name} (optional) for example ‘\textit{Cantharis vericatoria}’ is the scientific name of cantharidin;
\item \textbf{Other Name} (optional) for example ‘Spanish fly’ is a colloquial term for cantharidin;
\item \textbf{Other Similar} (optional) a list of similar substances that don’t have their own entries; for example, ‘choline’ is listed as similar to aspirin;
\item \textbf{Toxicity} a number from 1 (low) to 6 (highly toxic).\footnote{The toxicity level relates to the amount required for a lethal dose. It’s not a measure of symptoms. A level 6 toxin can cause death from a pinprick amount with few symptoms, whereas a level 2 toxin requires a much higher dose to kill but can have debilitating long-term symptoms from a non-lethal dose.}
\end{itemize}

\begin{itemize}
\item \textbf{Form/Deadly Parts} plants have a deadly parts item (identifying which part of the plant is toxic), while other entries list the form (for example, liquid or gas) that the toxin takes;
\item \textbf{Effects and Symptoms} such as headache or nausea;
\item \textbf{Reaction Time} how long for symptoms to occur;
\item \textbf{Antidotes and Treatments} whether or not an antidote is available or if there is known treatment;
\item \textbf{Notes} (optional) some additional information, sometimes including a case history.
\end{itemize}

I don’t want to bog down the examples with unrelated style code, so I’m just going to use the standard \texttt{description} environment. The entry data itself will also be significantly pared down to the following, which will correspond to glossary entry fields (custom ones will need to be defined):

\begin{itemize}
\item \texttt{name} the toxin name (as it will appear in the section title);
\item \texttt{description} information about the toxin;
\item \texttt{toxicity} (custom field) a number;
\item \texttt{method} (custom field) the method of administration, a list of one or more elements from the set: breathed, injected, membrane absorption, skin absorption, smoked, swallowed.
\item \texttt{symptom} (custom field) the symptoms, which should be a list of one or more elements from any of the symptom classification subsets, including: vital signs: \texttt{bradycardia, hyperthermia, hypertension, hypotension, hypothermia, tachycardia};
\item \texttt{head, eyes, nose, throat}: bad/unusual taste, etc.
\end{itemize}

(There are too many to list individually here.) I’ve omitted form (aerosol etc.) as the ‘Ordered by Form’ list can be achieved in the same way as the ‘Ordered by Method’ list. Similarly for the reaction time, which could be implemented with a numeric identifier like the toxicity.

2 Manual method

I’m going to start with an example document that doesn’t use the \texttt{glossaries} package. Since we live in a digital age where some people prefer to read books on their devices, I’ve used the \texttt{hyperref} package [2]. The chapters are numbered in the main matter, but sections aren’t numbered anywhere. The sections and subsections are too numerous to list in the table
of contents, but they would be useful in the PDF bookmarks. This can be achieved by setting the \texttt{tocdepth} counter to 0 (to only show chapters in the table of contents), setting the \texttt{secnumdepth} counter to 0 (to only show numbers for chapters) and using \texttt{hyperref}'s \texttt{bookmarksdepth} option to set the depth for the PDF bookmarks.

The \texttt{hyperref} package automatically creates an anchor at the start of each page where the anchor name is obtained from the formatted page number. The \texttt{frontmatter} command resets the \texttt{page} counter to 0 and changes the page number format to lowercase Roman numerals. Thus, the first page of the table of contents is 'i' and so the anchor for that page is \texttt{page.i}. The \texttt{mainmatter} command similarly resets the \texttt{page} and changes the format to arabic (the default page number format) so the anchor for the first page of the first chapter will be \texttt{page.1}.

Unfortunately, the title page (and its reverse) also use the default number format so, even though the numbering may be hidden by the empty page style, the page anchor is still created (\texttt{page.1} for the title page and \texttt{page.2} for its reverse). Since the page numbers are hidden, the simplest solution is to select a different number format that isn't used for any of the other pages. In this case, I've used the \texttt{alph} format. This means the first two pages have the anchors \texttt{page.a} and \texttt{page.b}. They're not required anywhere in the document but this prevents a conflict and ensures that any references to pages 1 or 2 in the index or glossaries (once they are added) link to the correct page.

The ‘Ordered by’ lists mostly have very narrow columns so I’ve used the \texttt{multicol} package [1]. As we’ll see, the ‘Ordered by Toxicity’ listing rounds 4.5 down to 4, which is what Stevens and Bannon do (although their toxicity list is in the opposite order).

Some of the scientific names are New Latin names, so I’ve provided a semantic command to typeset them:

\begin{verbatim}
\newcommand*{\latinname}[1]{\emph{#1}}
\end{verbatim}

This example document contains only one toxin (ammonia), but it’s already lengthy as it has a large number of symptoms.

\begin{verbatim}
\documentclass{book}
\usepackage{multicol}
\usepackage[bookmarksdepth=2]{hyperref}
\title{A Little Book of Poisons}
\author{Ann Author}
\setcounter{secnumdepth}{0}
\setcounter{tocdepth}{0}
\newcommand*{\latinname}[1]{\emph{#1}}
\begin{document}
\maketitle
\frontmatter\pagestyle{headings}
\tableofcontents
\mainmatter
\chapter{Household Poisons}
\section{Chemicals}
\subsection{Ammonia}
\begin{description}
\item[Other] Ammonium hydroxide.
\item[Toxicity] 4.5
\item[Method] Breathed.
\item[Symptoms] Tachycardia, blindness, lip/mouth irritation, burns, flushing, coughing, pulmonary edema, abdominal or stomach pain, restlessness, collapse, and pain.
\item[Description] Some information about ammonia.
\end{description}
\backmatter
\chapter{Ordered by Administration}
\begin{multicols}{3}
\section{Breathed}
\begin{itemize}
\item Ammonia
\end{itemize}
\end{multicols}
\chapter{Ordered by Symptoms}
\begin{multicols}{3}
\section{Vital Signs}
\subsection{Tachycardia}
\begin{itemize}
\item Ammonia
\end{itemize}
\% Lots of other sections omitted for brevity
\end{multicols}
\chapter{Ordered by Toxicity}
\begin{multicols}{3}
\section{Toxicity Rating 6}
\section{Toxicity Rating 5}
\section{Toxicity Rating 4}
\begin{itemize}
\item Ammonia
\end{itemize}
\section{Toxicity Rating 3}
\section{Toxicity Rating 2}
\end{multicols}
\end{document}
\end{verbatim}

3 Standalone entries

For this next example, I’m going to consider a cut-down version of the main matter in order to illustrate standalone entries. The simplest approach can be achieved with the base \texttt{glossaries} package, although this has limitations. For now, each entry just has a name, toxicity and description. The toxicity could be stored in one of the custom keys, such as \texttt{user1}, but I’ve decided to define a new key called \texttt{toxicity}:

\begin{verbatim}
\glsaddstoragekey{toxicity}{}
\end{verbatim}

\texttt{\bib2gls}: Standalone entries and repeated lists (a little book of poisons)
This both defines the toxicity key and provides a command called \texttt{toxicity} to access the value.

Since the descriptions are likely to be quite lengthy and may contain paragraph breaks, they are best defined with \texttt{longnewglossaryentry}:
\begin{verbatim}
\longnewglossaryentry{ammonia}
  \{name=ammonia,toxicity=4.5\}
  \{Some information about ammonia.\}
\longnewglossaryentry{nutmeg}
  \{name=nutmeg,toxicity=3\}
  \{Some information about nutmeg.\}
\longnewglossaryentry{lsd}
  \{name=LSD,toxicity=2\}
  \{Some information about LSD that includes a reference to nutmeg.\}
\longnewglossaryentry{botulinum}
  \{name=botulinum,toxicity=6\}
  \{Some information about botulism.\}
\end{verbatim}

These are all defined in the file \texttt{toxins.tex}, which needs to be input in the preamble (with either \texttt{\input} or \texttt{\loadglsentries}).

Since all the sections follow a set format, I'll define a command (\texttt{\toxin}) that simply takes a label and displays the complete section. To accommodate the mixture of sections and subsections, we have an associated command (\texttt{\toxinsection}) that can be redefined at the start of a chapter where necessary:
\begin{verbatim}
\newcommand{\toxinsection}{\section}
\newcommand{\toxin}[1]{\toxinsection{\glsentrytitlecase{#1}{name}}
  \begin{description}
    \item[Toxicity] \toxicity{#1}
    \item[Description] \glsentrydesc{#1}
  \end{description}
}
\end{verbatim}

This converts the name field to title case in the section heading using \texttt{\glsentrytitlecase}, which internally uses \texttt{\capitalisewords} provided by the \texttt{mfirstuc} package.

When writing in English, words such as ‘and’ should only be capitalized when they occur at the start of the title. Since such exceptions are language dependent, they aren’t implemented by default. The \texttt{mfirstuc-english} package provides the common English exceptions:
\begin{verbatim}
\usepackage{mfirstuc-english}
\end{verbatim}

This doesn’t affect the document so far, but it will later when the ‘order by symptoms’ list is added.

The main matter is now much shorter:
\begin{verbatim}
\chapter{Household Poisons}
\renewcommand{\toxinsection}{\section}
\section{Chemicals}
\toxin{ammonia}
\section{Food Poisoning}
\toxin{botulinum}
\chapter{Plants}
\chapter{Street Drugs}
\toxin{lsd}
\end{verbatim}

When just considering the main matter, this doesn’t seem like a significant improvement to the first example. It is easier to move the sections around, but the title case-changing can’t be implemented in the PDF bookmarks. So \texttt{\glsentrytitlecase} will expand to the original lowercase value in the bookmark.

It would be useful if the nutmeg reference in the LSD description had a hyperlink to the nutmeg section (created with \texttt{hyperref}). Such hyperlinks are normally achieved with the \texttt{glossaries} package using commands like \texttt{\gls}. However, the target anchor is typically in the glossary (implemented by the glossary style), but there isn’t a glossary in this document.

The \texttt{glossaries-extra} package provides a solution where you can use \texttt{\glsxtrglossentry}:
\begin{verbatim}
\glsxtrglossentry{#1}
\end{verbatim}

This command expands to just \texttt{\glsentryname} for the PDF bookmark, so there’s no difference in this respect. However, within the document text, this command creates the hypertarget and displays the name in the same way as the glossary styles. So I can adjust the case using the \texttt{glossname} attribute:
\begin{verbatim}
\glssetcategoryattribute{general}{glossname}
\end{verbatim}

This only applies to the section title on the page, not in the bookmarks.

The LSD entry can now be modified to include a hyperlink to the nutmeg section:
\begin{verbatim}
\longnewglossaryentry{lsd}
  \{name=LSD,toxicity=2\}
  \{Some information about LSD that includes a reference to \gls{nutmeg}.\}
\end{verbatim}

There are over two hundred toxins listed in the book. At the moment, all my definitions are stored in my \texttt{toxins.tex} file, but I could store them in a \texttt{bib} file instead. This would make it easier to share the data across multiple documents. For example, the \texttt{bib} file may include brief summaries that can be used as a description in other shorter documents as well as the long description for this catalogue of toxins. For example:
\begin{verbatim}
@entry{nutmeg,name={nutmeg},
  summary={Short description of nutmeg.},
  longdescription={Long description of nutmeg.}}
\end{verbatim}

The document then can choose the appropriate field for the description using field aliases. I’m not going to do this here in order to keep the examples as simple as possible.

Nicola L. C. Talbot
My toxins.tex file can easily be converted to toxins.bib using convertgls2bib:

```
convertgls2bib toxins.tex toxins.bib
```

Then I need to replace the code that inputs toxins.tex with:

```
\GlsXtrLoadResources[src=toxins]
```

and add the record package option.

The \latinname command can be provided in the preamble of the bib file to ensure that it’s defined:

```
@preamble{"
\providecommand*{\latinname}[1]{\emph{#1}}"
```

Again, this is something that’s useful if the bib file is shared with other documents, but isn’t essential for this example as the command is already defined in the document preamble.

Normally, bib2gls will select entries from the bib file if they have records in the aux file (which are created with commands like \gls or \glsadd) or if they depend on selected entries; for example, nutmeg needs to be selected if LSD is selected, since the LSD entry depends on the nutmeg entry.

In this case, though, bib2gls doesn’t select any entries because nothing creates a record. (The \gls command in the LSD description will only create a record if the LSD entry is selected and has its description expanded in the document, but there are no LSD records, so LSD won’t be selected.)

I could instruct bib2gls to select all entries, but some entries may need to be omitted. For example, the publisher may decide that the print cost for the physical edition is too large, so some entries may need to be dropped.

One approach is to use \glsadd in the definition of \toxin:

```
\newcommand*{\toxin}[1]{%
    \glsadd{#1}\% index this entry
    \toxinsection{\glsstrglossentry{#1}}
    \begin{description}
    \item[Toxicity] \toxicity{#1}
    \item[Description] \glsentrydesc{#1}
    \end{description}
}
```

This ensures that each entry listed in the book will be selected by bib2gls.

The problem of the case- conversion for PDF bookmarks can now be solved as there are some resource options that instruct bib2gls to change the case of field values:

```
name-case-change=title
```

However, this will cause \gls{nutmeg} to start with a capital unless the text field is set to the original value. This can be done with:

```
replicate-fields={name=text}
```

This will copy the value of the name field into the text field. (If the target field is already set, the default behaviour is to leave it unchanged.) Replication is always performed before case-changing, regardless of the resource option ordering. If the source field (name in this case) is not set, the default is to do nothing. But in this case, I want to obtain the value from the fallback if name is missing:

```
replicate-missing-field-action=fallback
```

This means I can now dispense with the glossname attribute.

## 4 Comma-separated list fields

So far I haven’t included the method and symptoms in my entry definitions. I can define two more custom keys in the same way as for toxicity:

```
\glsaddstoragekey{method}{}{method}
\glsaddstoragekey{symptom}{}{symptom}
```

I could simply set the values to free-form text:

```
@entry{ammonia, name={ammonia},
    toxicity={4.5},
    description={Some information about ammonia.},
    method={Breathed.},
    symptom={Tachycardia, blindness, lip/mouth irritation, burns, flushing, coughing, pulmonary edema, abdominal or stomach pain, restlessness, collapse, and pain.}
}
```

However, I decided to adopt a different approach. First, I created a file called methods.bib containing:

```
@index{breathed}
@index{injected}
@index{membraneabsorption, name={membrane absorption}}
@index{skinabsorption, name={skin absorption}}
@index{smoked}
@index{swallowed}
```

The method fields are all going to be comma-separated lists of the method entry labels.

The symptoms are defined in a similar way (in a file called symptoms.bib) but they have an unknown topic field, which will be ignored by bib2gls unless it is aliased or defined in the document (see later). For example:

```
@index{hyperthermia, name={fever\MFUwordbreak{\slash}hyperthermia},
    text={fever},
    topic={vital signs}}
```

The method fields are all going to be comma-separated lists of the method entry labels.

The symptoms are defined in a similar way (in a file called symptoms.bib) but they have an unknown topic field, which will be ignored by bib2gls unless it is aliased or defined in the document (see later). For example:

```
@index{hyperthermia, name={low body temperature\MFUwordbreak
    {\slash}hypothermia},
    text={hypothermia},
```

bib2gls: Standalone entries and repeated lists (a little book of poisons)
Note that while most of these are defined using \texttt{index} there are some defined with \texttt{indexplural}. These entries will default to having the categories set to ‘index’ and ‘indexplural’. This will cause complications later, so all entries will be assigned to the ‘general’ category with the resource option: \texttt{category=general}

The names will be converted to title case so the slash needs to be marked up as a word break using \texttt{\textbackslash MFUwordbreak}, otherwise the word following the slash won’t have its case changed. You may prefer to define a string:
\begin{verbatim}
@string{SLASH="\textbackslash MFUwordbreak{\slash}"}
\end{verbatim}
and use string concatenation:
\begin{verbatim}
name={fever} # SLASH # {hyperthermia},
\end{verbatim}

The ammonia entry can be defined as:
\begin{verbatim}
@entry{ammonia,name={ammonia},
  toxicity = {4.5},
  description={Some information about ammonia.},
  method = {breathed},
  symptom = {tachycardia, blindness, mouth irritation, burn, flushing, coughing, pulmonary edema, abdominal, restlessness, collapse, pain}
}
\end{verbatim}

The line breaks in the comma-separated lists above can be problematic since these lists will internally be passed to \texttt{\textbackslash for} in the document, but it is possible to get \texttt{bib2gls} to strip the whitespace, if you’d rather not omit them.

There are two options, \texttt{labelify} and \texttt{labelify-list}, that can be used to strip any content that can’t occur in a label. The former is intended for fields containing a single label and the latter is for fields containing a comma-separated list of labels. Both are governed by \texttt{labelify-replace}, so the following can be used to strip any whitespace:
\begin{verbatim}
labelify-list={method,symptom},
labelify-replace={{\string s+}{}{}}
\end{verbatim}

This means that you can introduce extra space in the \texttt{bib} file to make it more readable. Further, since this option also automatically removes empty items, it’s also possible to replace \texttt{\textbackslash and} with a comma:
\begin{verbatim}
labelify-list={method,symptom},
labelify-replace={\textbackslash string\textbackslash s+{}{},},
\end{verbatim}

This means that the list ‘A and B’ becomes ‘A,B’. The list ‘A, B, and C’ becomes ‘A,B,C’; the empty element is then stripped, leaving ‘A,B,C’.

So the \texttt{symptom} field can be set as:
\begin{verbatim}
symptom = {tachycardia, blindness, mouth irritation, burn, flushing, coughing, pulmonary edema, abdominal, restlessness, collapse, pain}
\end{verbatim}

This is not only easier to read but also makes it suitable for use without the \texttt{symptoms.bib} file.

The \texttt{\textbackslash gllseelelist} command (provided by the base \texttt{glossaries} package) formats a comma-separated list of entry labels. This was designed for the use of cross-referencing with the \texttt{see} field [5], but may be used with any list of entry labels. If you want to ensure that the argument is fully expanded, use \texttt{glossaries-extra’s} \texttt{\textbackslash gllseelelist} instead (which internally uses \texttt{\textbackslash gllseelelist}).

The \texttt{\textbackslash toxin} command can be modified to include formatted lists of symptoms, but \texttt{\textbackslash gllseelelist} doesn’t index so the method and symptom entries won’t be selected. In order to ensure that they are selected, \texttt{bib2gls} needs to be told that the \texttt{method} and \texttt{symptom} fields contain lists of dependent entries:
\begin{verbatim}
dependency-fields={method,symptom}
\end{verbatim}

The method and symptom entries don’t have any targets (at the moment) so the hyperlinks need to be suppressed. Also the \texttt{name} field has had a case-conversion applied. Both problems can be fixed by redefining \texttt{\gllseeitem} to just use \texttt{\gllseelelist}: \texttt{\renewcommand*{\gllseeitem}[1]{\gllseelelist{#1}}}

If you want the first item capitalised you can redefine \texttt{\gllseefirstname}:
\begin{verbatim}
\renewcommand*{\gllseefirstname}[1]{\Gllseelelist{#1}}
\end{verbatim}
The separator between the last two items in the list is given by \texttt{\gllseefirstsep}, which defaults to \texttt{\textbackslash and}. If you want to change this to use ‘and’ instead:
\begin{verbatim}
\renewcommand*{\gllseefirstsep}{ and }
\end{verbatim}
If your preference is for an Oxford comma you will also need:
\begin{verbatim}
\renewcommand*{\gllseeoxyfordsep}{, and }
\end{verbatim}
This may seem a bit redundant since the end result is much the same as the original field value, but the hyperlink will be added in a later example once the corresponding lists have been created.

If you want the \texttt{method} and \texttt{symptom} elements to be alphabetically ordered, then you can instruct \texttt{bib2gls} to do this with the \texttt{sort-label-list} option:
\begin{verbatim}
\renewcommand*{\gllseelelist}{\gllseelelist{\textbackslash {method,symptom}:en:glseelelist}}
\end{verbatim}
This indicates that the method and symptom fields are comma-separated lists and that bib2gls should reorder these lists according to the en sort method (English) where the sort value is obtained by encapsulating the list element with \glsentryname (which bib2gls recognises). This ensures that the list is ordered by the displayed name rather than the label.

5 The index

Since bib2gls sorts by default, a convenient side-effect is that the index can easily be added at the end of the document using the bookindex glossary style (which doesn't show descriptions). As with the other provided glossary styles, this will create a hypertarget, which will cause a conflict, but this can be switched off with the target=false option:
\printunsrtglossary[target=false,title=Index]
The bookindex style isn't loaded by default, so you'll also need to specify it explicitly:
\usepackage[record=nameref,stylemods=bookindex,style=bookindex]{glossaries-extra}

I've set the style as a package option as the other glossaries discussed later will also use this style. If you want letter groups, remember to use the \--group (or \-g) switch when you invoke bib2gls.

If any entries have the see, seealso or alias fields set, \glsseeitem will need to be restored to its original value for the index. The simplest way to do this is to localise the redefinition. So instead of redefining it in the preamble, it can be redefined within a scoped context within the definition of \toxin. Since environments automatically add scoping, the redefinition can be placed inside the description environment.

Some of the entry descriptions may span multiple pages, in which case you may prefer to have a page range in the index. This can be achieved with explicit location ranges. The position of \glsadd also needs an adjustment. This command switches to horizontal mode (as complications can occur in certain situations otherwise), which means that the page number could be off if the section heading is moved to the start of the next page. If \glsadd is placed after the heading then it will cause an unwanted space before the start of the description environment. The best solution is to place it in the section title and use the optional argument for the bookmark.
\newcommand*{\toxin}[1]{%        
  \ifglsentryname{}{%          
    \glsadd[format={}]{#1}\begin{description}        
      \let\glsseeitem\glsentrytext
      \item[Toxicity] \toxicity{#1}\end{description}
  \fi}
\newcommand*{\filteremptylocation}[1]{%  
  \glsentrytext\glsadd[format=){}]{#1}\end{description}
}

There is a problem with the method and symptom entries. They haven't been indexed anywhere in the document so they don't have a page list. This could be solved by redefining \glsseeitem to use \glistext instead of \glsentrytext, but this increases the complexity of the document build and could lead to lengthy page lists in the index, especially for common methods (such as swallowing, which applies to most toxins) or symptoms (there's a fairly sizable list for convulsions). Since the final version of this example will have lists of methods and symptoms, there's no need for them to appear in the index.

There are two basic approaches to removing entries from a list: put them in a different glossary or filter them when displaying the list. The first approach can be a bit tricky if all entries are being processed by a single resource command. One way would be to rename toxins.bib to main.bib and use the resource option:
\texttt{type=\texttt{(same as base)}}

This will set the type field to the file basename, without the bib extension. So the entries defined in main.bib will be assigned to the default main glossary, the entries defined in methods.bib will be assigned to the glossary identified by the label methods (which will need to be defined), and similarly entries defined in symptoms.bib will be assigned to a user-provided symptoms glossary.

The second approach keeps all the entries in one glossary but uses the hook that's provided to help skip entries. This is discussed in more detail in a previous article [5], but essentially, in order to avoid problems involved in using iterative code within a tabular-like environment (which some glossary styles use), the entries are first iterated over outside of the glossary and the glossary contents are appended to an internal control sequence. There's a hook that's used in this stage which can skip the current iteration to prevent an entry from being appended.

I've defined a custom command to filter entries with empty locations because it may be useful for other lists (either in this document or placed in a package for the use of other documents):
\newcommand*{\filteremptylocation}[1]{%        
  \glsentrytext\glsadd*{location}{#1}
The process hook will be `%\let` to this command.

Although each entry in the index has a location list, it might be useful to have the entry name as a hyperlink to its section in the main part of the document. The `bookindex` style provides a command that’s used to format the entry name, which takes the entry label as its argument. Again I’m defining a custom command which the style command can locally be `%\let` to. This simply encapsulates the name with a hyperlink:

```latex
\newcommand*{\linkedbookname}[1]{\glshyperlink{\glossentryname{#1}}{#1}}
```

The starred form of `%\printunsrtglossary` has a mandatory argument where the code to initialise the hooks can be placed. This is scoped so it won’t alter any subsequent lists.

```latex
\printunsrtglossary*[target=false,title=Index]{%\let\printunsrtglossaryentryprocesshook\filteremptylocation%\let\glsxtrbookindexname\linkedbookname}
```

Unfortunately the index now has terms in title case, which doesn’t look quite right. I used `name-case-change` to remove the non-expandable case-changing from the PDF bookmarks, but this has now had an unwanted side-effect. To overcome this problem, I can create a field to store the bookmark title:

```latex
\glsaddstoragekey{bookmark}{}{\pdfname}
```

Now, instead of copying the `name` to the `text` field, I copy it to this new `bookmark` field:

```latex
\glsaddstoragekey{bookmark}{}{\pdfname}
```

and instead of `name-case-change` I now need to use:

```latex
field-case-change={bookmark=title}
```

Any formatting commands can be stripped by instructing `bib2gls` to interpret the `bookmark` field:

```latex
interpret-fields={bookmark}
```

Alternatively, the `\pdftstringdefDisableCommands` command from `hyperref` can be used to discard problematic tokens.

The `%\toxin` command needs to be modified to use this field:

```latex
\toxinsection{\pdfname[#1]}\glsentrysection{#1}\glsaddformat={#1}\glsunsetcategoryattribute{general}{glossname}
```

6 Synonyms and related terms

The scientific names, alternative names and related substances could be added in a similar way to the symptoms and methods, but it would be useful to have these terms in the index.

There are three cross-referencing fields available [5]: `see`, `seealso` and `alias`. The first two take a comma-separated list of labels. The `alias` field can only have a single label as the value.

I’m going to use the `alias` field for the scientific name, the `see` field for alternative names (‘Other’) and the `seealso` field for similar substances (‘Related’). For example:

```latex
@index{cbot, name={\latinname{Clostridium botulinum}}, alias={botulinum}}
@index{botulism, see={botulinum}}
@index{botox, see={botulinum}}
@index{lsd-long, name={\textit{lsd}}, alias={lsd}}
@index{lysergide, see={lsd}}
@index{ammoniumhydroxide, name={ammonium hydroxide}, alias={ammonia}}
```

The scientific name for nutmeg is `Myristica fragans`:

```latex
@index{mfragans, name={\latinname{Myristica fragans}}, alias={nutmeg}}
```

But some other nutmeg species are also listed:

```latex
@index{margentea, name={\latinname{Myristica argentea}}, alias={nutmeg}}
@index{mmalabarcia, name={\latinname{Myristica malabarcia}}, alias={nutmeg}}
```

This complicates things a little as they need their common name as well:

```latex
@index{Papuan-nutmeg, name={Papuan nutmeg}, see={margentea}}
@index{Bombay-nutmeg, name={Bombay nutmeg}, see={mmalabarcia}}
```

These new entries aren’t referenced anywhere in the document, nor are the selected entries dependent on them, so I need to change the selection criteria to include entries that cross-reference the selected entries:

```latex
selection={recorded and deps and see}
```

The `%\toxin` command needs to be adjusted so that it shows the other names. Entries only have fields that store dependent entries (`see`, etc.), not

Nicola L. C. Talbot
the reverse. Whilst it is possible to iterate over all entries to find the synonyms, it’s not very efficient.

The resource options save-from-alias, save-from-see and save-from-seealso provide a solution. These define, respectively, the fields from-alias, from-see and from-seealso that contain the required information. The `\toxin` command now needs to check if any of these fields have been defined. For example:

```
\glsxtrifhasfield*{from-see}{#1}{\item[Other]
\glsxtrseelist\glscurrentfieldvalue}{}
```

The `from-alias` field could be dealt with in the same way, but the common names of the other nutmeg species won’t show.

A simple solution is to define a command that checks the `from-see` field that can be used to encapsulate the items in the list:

```
\newcommand{\seemandother}[1]{\% \glsxtrifhasfield*{from-see}{#1}{\item[Other]
\glsxtrseelist\glscurrentfieldvalue}{\glsentrytext{#1}}}
```

and also an analogous command `\Seemandother` that uses `\Glsentrytext` instead, used for the first item in the list. There’s a custom command to switch to these commands and display the list:

```
% \toxinitemlist{label}{field}{title}
\newcommand{\toxinitemlist}[3]{% \glsxtrifhasfield*{#2}{#1}{\item[\#1]
\glsxtrifhasfield*{#3}{#1}{\glsentrytext{#1}}}{\glsentrytext{#1}}}
```

I’ve made an inner command to make it easier to adjust the actual formatting:

```
\newcommand{\formattoxinitemlist}[2]{% \glsxtrifhasfield*{#1}{#2}{\item[\#1]
\glsxtrifhasfield*{#2}{#1}{\glsentrytext{#1}}}{\glsentrytext{#1}}}
```

The description environment within the `\toxin` definition can now be written in a more compact form:

```
\begin{description}
\toxinitemlist{#1}{from-alias}{Scientific Name}
\toxinitemlist{#1}{from-see}{Other}
\toxinitemlist{#1}{from-seealso}{Related}
\end{description}
```

The `\booklinkname` command is now going to cause a problem in the index as these new cross-reference terms don’t have a target. There are a number of ways around this. For example, a conditional can be added to use a hyperlink only if the toxicity field is set:

```
\newcommand{\linkedbookname}[1]{% \glsxtrifhasfield{toxicity}{#1}{\glshyperlink{\glossentryname{#1}}{#1}}{}
\glossentryname{#1}}
```

It would, however, be more convenient if the other names could have a hyperlink to their main entry. This could be done by consulting the `alias`, `see` and `seealso` fields in turn, which leads to a complicated set of nested conditionals and also doesn’t work for Bombay nutmeg and Papuan nutmeg.

The save-crossref-tail resource option is useful here as it will save the tail label from a cross-reference trail in the `crossref-tail` field. This requires only one extra conditional:

```
\newcommand{\linkedbookname}[1]{% \glsxtrifhasfield{toxicity}{#1}{\glshyperlink{\glossentryname{#1}}{#1}}{% \glsxtrifhasfield{crossref-tail}{#1}{\glshyperlink{\glossentryname{#1}}{\glscurrentfieldvalue}}{% \glossentryname{#1}}}}
```

7 Order by toxicity

The glossaries package allows multiple glossaries. A default one is provided with the label `main`. When a new glossary is defined, an internal command is constructed from the name `glolist@type` where `<type>` is the glossary label. When a new entry is defined, its label is appended to the glossary’s internal list command’s replacement text with a comma separator. It’s this list that `\printunsrtglossary` iterates over.

The glossaries-extra package provides a command that copies an entry label to another glossary list. This means that the entry is only defined once and its type field is set to its original glossary (this can be considered the entry’s primary glossary) but the entry will also appear in the other glossary’s list. (This approach can’t be used with `makeindex` or `xindy`.)

For example:

```
\newglossaryentry{sample}{name=sample,description={}}
\newglossary{another}{Another}
\glsxtrcopytoglossary{sample}{another}
\begin{document}
\printunsrtglossaries
\end{document}
```
This will display two glossaries, both containing the sample entry. This method allows a duplicate list, which may have a different order, without the overhead of duplicate entry definitions, and it’s this method that’s employed with \texttt{bib2gls}'s secondary resource option.

The syntax for this option is:

\begin{verbatim}
  secondary=\{sort\}::\{field\}::\{type\}
\end{verbatim}

where \{sort\} indicates the sort method and \{type\} is the glossary label.

The \{field\} part is optional and indicates the field to use for sorting. The previous article \cite{7} discussed sorting and, in particular, the system of fallbacks used to determine the value used for comparison if the \texttt{sort} field isn’t set.

A by-product of the sorting function (regardless of the field used for sorting) is that it assigns the actual sorting value (possibly obtained from fallbacks, with word breaks marked, suffixes appended etc.) to the \texttt{sort} field. This means that if you want a secondary glossary, you will need to choose a different field for the sort value unless you want to reuse the same sort values from the primary sort (which would usually be redundant). The secondary sort method will store its actual sort value in the internal field \texttt{secondarysort} to avoid conflict, in the event that you need to access both values in your document.

I’ve used the \texttt{--group} switch to add letter groups to my index. You may recall from previous articles that this will store the group label (obtained as a by-product of sorting) in the \texttt{group} field. To avoid conflict the secondary sort function will store the group label in the \texttt{secondarygroup} internal field, and \texttt{bib2gls} will append the required redefinition to the secondary glossary’s preamble so that it will automatically switch to the \texttt{secondarygroup} field.

So to have a secondary glossary ordered according to the \texttt{toxicity} field (from highest to lowest):

\begin{verbatim}
  secondary=integer-reverse::toxicity::bytoxicity
\end{verbatim}

Here I’ve indicated that the secondary glossary has the label \texttt{bytoxicity}. If you inspect the \texttt{glistex} file, you should find the line:

\begin{verbatim}
  \providecommand{\bibglsnumbergrouptitle}{%\par
  \glsxtrsetgrouptitle {\bibglsnumbergrouptitle#1}}
\end{verbatim}

This command is provided at the start of the file:

\begin{verbatim}
  \providecommand{\bibglsnumbergrouptitle}{%\par
  \glsxtrsetgrouptitle {\bibglsnumbergrouptitle#1}}
\end{verbatim}

The group label is obtained from the control sequence \texttt{\bibglsnumbergroup} (which must fully expand). This command is also provided:

\begin{verbatim}
  \providecommand{\bibglsnumbergrouptitle}{%\par}
\end{verbatim}

It’s this definition that causes all entries that have been sorted numerically to be placed in the ‘Numbers’ group. For this example, I’d like the groups to correspond to the toxicity levels, so I need to define this command before the \texttt{glistex} file is input:

\begin{verbatim}
  \newcommand{\bibglsnumbergrouptitle}{%\par}
\end{verbatim}

The corresponding title is obtained from a command that is also provided in the \texttt{glistex} file:

\begin{verbatim}
  \providecommand{\bibglsnumbergroup}{%\par}
\end{verbatim}

Again I can provide my own definition in the document to override this. For example:

\begin{verbatim}
  \newcommand{\bibglsnumbergrouptitle}{%\par}
\end{verbatim}

The \texttt{bookindex} headers are formatted according to:

\begin{verbatim}
  \glsxtrbookindexformatheader{(title)}
\end{verbatim}

This defaults to a centred format. I’ve decided to provide a different format, but I don’t want to apply it to the index as well, so the redefinition will need to be scoped.

The custom header formatting is quite simplistic for this example:

\begin{verbatim}
  \newcommand{\orderbyheader}{%\par}
\end{verbatim}

This can be adjusted as required.

As with the index, the list includes entries that don’t have the \texttt{toxicity} field set. These will end up with a sort value of 0 and can be filtered using a method similar to that employed for the index. However, it’s simpler to get \texttt{bib2gls} to filter them:

\begin{verbatim}
  secondary-not-match={toxicity={}}
\end{verbatim}

The glossary can be displayed in the usual way:

\begin{verbatim}
  \printunsrtglossary[type=bytoxicity, target=false]
\end{verbatim}

Nicola L. C. Talbot
This will include the page list for each entry, which you may prefer to omit. The nonumberlist option can be added to suppress it. On the other hand, it can be useful to have a way to link back to the main definition. As with the index, a hyperlink can be added to the name using the same method as earlier. This is useful for the reader of the PDF version, but for the paperback version it might be helpful to just list the primary page number (rather than the complete list). For this, I’ve defined a custom location format command:

\newcommand{\mainfmt}[1]{\glsnumberformat{#1}}

This uses the default formatting, but I can use it to identify the principal page in my custom $\texttt{\textcolor{blue}{\texttt{toxin}}}$ command:

\begin{verbatim}
\section[\pdfname{#1}]{\glsxtrglossentry{#1}\
\glsadd[format=mainfmt]{#1}\
\glsadd[format=]{#1}}
\end{verbatim}

This primary location format $\texttt{\textcolor{blue}{\texttt{mainfmt}}}$ is identified with:

\begin{verbatim}
primary-location-formats=mainfmt
\end{verbatim}

I can instruct \texttt{bib2gls} to move the primary locations out of the normal location list and into a field called primarylocations:

\begin{verbatim}
save-primary-locations=remove
\end{verbatim}

To ensure that $\texttt{\textcolor{blue}{\texttt{printunsrtglossary}}}$ uses this field for the location list:

\begin{verbatim}
\renewcommand{\GlsXtrLocationField}{primarylocations}
\end{verbatim}

To prevent the primary locations from being merged with the explicit range formation:

\texttt{bib2gls} -\texttt{g} --retain-formats mainfmt toxinbook

where the document is in the file $\texttt{\textcolor{blue}{\texttt{toxinbook.tex}}}$.

\section{Order by method}

The method list is superficially similar. It’s not possible to use multiple secondary options in one resource command, but it’s possible to have a second resource set that copies entries to another glossary. First define the new glossary:

\begin{verbatim}
\newglossary*{bymethod}{By Method}
\end{verbatim}

Now for the second resource command:

\begin{verbatim}
\GlsXtrLoadResources[
src=methods,type=bymethod,
selection={selected before},
action=copy
]
\end{verbatim}

This selects all the entries in $\texttt{\textcolor{blue}{\texttt{methods.bib}}}$ that were previously selected, sorts them and copies their labels to the $\texttt{\textcolor{blue}{\texttt{bymethod}}}$ glossary. Internally (that is, within \texttt{bib2gls}’s Java code) a new object representing each entry is created with the information obtained by reparsing the \texttt{\textcolor{blue}{\texttt{bib}}} file. So any modifications made by the previous resource set won’t be present in this resource set (unless the modifications are repeated). This means that the sort field will be missing again: the value won’t be retained from the previous resource set. However, from \LaTeX’s point of view, each entry is defined once.

This now provides a target for the methods, so the method list in $\texttt{\textcolor{blue}{\texttt{\textcolor{blue}{\texttt{toxin}}}}}$ can have a hyperlink for each entry:

\begin{verbatim}
\renewcommand*{\glsseeitem}[1]{\glshyperlink{\glsentrytext{#1}}{#1}}
\renewcommand*{\glsseefirstitem}[1]{\glshyperlink{\Glsentrytext{#1}}{#1}}
\end{verbatim}

So far this just lists the methods. The sub-list of relevant toxin entries needs to follow each method name. This looks like a hierarchical glossary but it has child entries with multiple parents. The structure is essentially a set of nested glossaries with an outer glossary (the $\texttt{\textcolor{blue}{\texttt{bymethod}}}$ glossary) where each element is followed by an inner glossary.

The glossary process hook can be used to create throwaway glossaries:

\begin{verbatim}
\let\printunsrtglossaryentryprocesshook\provideignoredglossary
\end{verbatim}

This creates a glossary with the same label as the entry, but the hook will have to be reverted before the inner glossaries are processed.

There’s another hook that’s used after processing and just before the glossary is displayed. This can be used to populate the throwaway glossaries and reset the process hook. First a command that does the action:

\begin{verbatim}
\newcommand{\populatemethods}{%\renewcommand\
\printunsrtglossaryentryprocesshook[1]{}%\forglsentries[main]{\thislabel}{{\glsxtrforcsvfield*{\thislabel}{method}{{\populatedo}}}%}
\end{verbatim}

with a list handler:

\begin{verbatim}
\newcommand{\populatedo}[1]{\glsxtrcopytoglossary{\thislabel}{#1}}
\end{verbatim}

Then the hook needs to be assigned to this command within a scoped context:

\begin{verbatim}
\let\printunsrtglossarypredoglossary\populatemethods
\end{verbatim}

The glossary entry handler needs to not only display the current entry (as it does by default with $\texttt{\textcolor{blue}{\texttt{\textcolor{blue}{\texttt{glsxstrforcsvfield}}}}}$) but also follow it with an inner glossary. First the custom nested handler that takes the current entry label as the argument:

\begin{verbatim}
\bib2gls: Standalone entries and repeated lists (a little book of poisons)
\end{verbatim}
This tests if there’s a glossary with a label matching the entry’s label and, if it exists, that glossary will be displayed (but without the title). As with the ordered by toxicity glossary, only the primary location is displayed and the name is hyperlinked to easily jump back to the main definition.

All glossaries are using the `bookindex` style, and since the method and toxin entries are all top-level entries (they don’t have a parent), they will all end up with their names formatted in the same way (as a top-level entry). The method entries need to have their names formatted in the same way as the group titles to be consistent with the order by toxicity glossary.

(Again, the style is simplistic to reduce the complexity of the example.) It’s then possible to switch to this in a scoped context:

\newcommand{\prenamesep}{\par}
\newcommand{\orderbyname}[1]{\orderbyheader{\glossentryname{#1}}
   \prenamesep}

9 Order by symptoms

The symptoms list is more complicated as it’s divided into different categories: `vital signs`, ‘head, eyes, ears, nose, throat’, ‘skin’, ‘heart’ and so on. These correspond to the custom `topic` field that has so far been ignored by bib2gls. Note that these topics aren’t listed in alphabetical order. Within each topic is a sub-list of symptoms, which is ordered alphabetically. I’m first going to start off with the topics alphabetically ordered and then make an adjustment to achieve the desired result.

The topics and their sub-lists are essentially hierarchical, so the `topic` field can be aliased to `parent`. The `labelify` option can strip the spaces using the same `labelify-replace` setting used earlier.

```
field-aliases={topic=parent},
labelify={parent}
```

The parent entries (representing the topics) need to be defined, so I’ve created a new file called `topics.bib` that contains:

```
@index{vitalsigns,name={vital signs}}
@index{head,
    name={head, eyes, ears, nose, throat}}
@index{skin}
@index{heart}
@index{airway,
    name={airway and lungs}}
@index{gastrointestinal,
    name={gastrointestinal system}}
@index{fluids,
    name={fluids and electrolytes}}
@index{neurological,
    name={neurological system}}
@index{psychiatric}
@index{wholebody,
    name={whole body and miscellaneous symptoms}}
```

This file needs to be added to the `src` list:

```
src={toxins,methods,symptoms,topics}
```

Since child entries depend on their parent, the parent entries will automatically be selected when the child entry is selected. The topics and symptoms can be copied to a new glossary in the same way as the methods:

```
\newglossary*{bysymptoms}{By Symptoms}
\GlsXtrLoadResources{src={symptoms,topics},
type=bysymptoms,
    selection={selected before},
    action=copy,
    field-aliases={topic=parent},
    labelify={parent},
    labelify-replace={{\string\s+}{}}}
```

The process hook is similar to the hook used for the method list but the throwaway glossaries are only created for child entries:

```
\newcommand{\symptomsprocesshook}[1]{\ifglishasparent{#1}{\provideignoredglossary{#1}}}
```

The code to populate the symptom glossaries is similar to that used for the methods:

```
\newcommand{\populatesymptoms}{\forglsentries{main}{\thislabel}}
```

Nicola L. C. Talbot
The entry handler is the same one as used before. The formatting of the sub-items (the symptom entries) needs adjusting. The `bookindex` style formats the sub-item names according to:
\glsxtrbookindexsubname{(label)}
The default definition uses \glsxtrbookindexname so this will need to be changed. First I need a custom sub-header to match the headers used in the toxicity and method lists:
\newcommand{\orderrysubheader}{\par\raggedright\bfseries #1\par}
Again this is simplistic, to be modified as required. The custom command for child entry names is:
\newcommand{\orderbychild}{\pdfbookmark[2]{\pdfname{#1}}{\glsxtrbookindexbookmarkprefix#1\orderrysubheader{\glossentryname{#1}}}\par}
I added the `mfirstuc-english` package earlier, but up until now it hasn’t been needed. This package can be implemented by `bib2gls` but isn’t by default. In order to ensure that the title-case word exceptions provided by that package are used by `bib2gls`, it’s necessary to use the `--packages` (or `-p`) switch:
bib2gls -g --retain-formats mainfmt \ --packages mfirstuc-english toxbook
Finally, I want to have the topics listed in the order that they are defined in the `topics.bib` file. This is quite awkward as it’s not possible to apply a different sort method to each hierarchical level. However, it is possible to encapsulate the sort value (after it has been obtained from fallbacks and any other processing, such as word breaks and suffixes). The encapsulation command must take two arguments: the first is the sort value that has been determined so far, and the second is the entry’s label.
It’s possible to save the entry definition index using `save-definition-index`. With this setting, the definition index can be accessed with the command \bibglsdefinitionindex. The aim here is to define a command that finds out if an entry has a parent. If it has, then the ordinary sort value is used. If it hasn’t, then the definition index is used instead. Since the sort method is alphabetical, the definition index will need to be zero-padded to ensure that it’s correctly ordered. Here I’ve padded up to six digits, which should be ample:
\bibgls: Standalone entries and repeated lists (a little book of poisons)
For the symptoms and method lists, the filter needs to be in the inner glossary:
\newcommand{\nestedhandler}[1]{% 
  \ifglossaryexists*{#1}{%
    \printunsrtnigglossary[type={#1},
      groups=false,target=false]
    
  \let\glsxtrbookindexname\linkedbookname
  \let\printunsrtglossaryentryprocesshook\filteremptylocation
  \renewcommand{\GlsXtrLocationField}{primarylocations}%
  
}{%%%}
%
}
%
%

Notice that the cross-reference fields only reference the primary entry, not the dual. This means that the scientific name and other names will be missing for the dual entry. However, it’s possible to adjust the definition of \toxinitemlist so that it fetches the information from the primary entry. In order to do this it’s first necessary to instruct \bib2gls to save the label of the opposite entry for dual entries. This can be done with the dual-field resource option, which will save the label of the opposite entry in the dual field. This means that the dual.botulinum entry will have the dual field set to botulinum and the botulinum entry will have the dual field set to dual.botulinum.
\newcommand{\toxinitemlist}[3]{% 
  \glsxtrifhasfield*{#2}{#1}{% 
    \formattoxinitemlist[#3]{\glscurrentfieldvalue}%
  }% 
  %
  \glsxtrifhasfield*{dual}{#1}{% 
    \% is a dual entry
    \glsxtrifhasfield*{#2}{% 
      \% 
      \formattoxinitemlist[#3]{\glscurrentfieldvalue}%
  }% 
  %
  %
  %
}
%
%

The complete document and \bib files can be downloaded [4]. I’ve used the uelem package and hyperref’s hidelinks option, so the hyperlinks show up underlined, to make them visible in the printed figures here. The key command for that:
\renewcommand{\glsxtrhyperlink}[2]{% 
  \hyperlink{#1}{\uline{#2}}}
%
%

\References
\begin{itemize}
  \item[4] N. Talbot. Sample \bib files.
dickimaw-books.com/latex/tugboat-bib2gls.
\end{itemize}

\textcopyright Nicola L. C. Talbot
School of Computing Sciences
University of East Anglia
Norwich Research Park
Norwich NR4 7TJ
United Kingdom

https://www.dickimaw-books.com
Chapter 1

Household Poisons

Chemicals

Ammonia

Scientific Name: Ammonium hydroxide.
Toxicity: 4.5
Method: Breathed
Symptoms: Abdominal or stomach pain, blindness, burns, collapse, coughing, flushing, mouth irritation, pain, pulmonary edema, restlessness, and tachycardia.
Description: Some information about ammonia.

Food Poisoning

Botulinum

Scientific Name: Clostridium botulinum.
Other: Botox and botulism.
Toxicity: 6
Method: Injected and swallowed
Symptoms: Blurred or double vision, nausea, and paralysis.
Description: Some information about foodborne botulism.

Chapter 2

Plants

Nutmeg

Scientific Name: Myristica argentea (Papuan nutmeg), Myristica fragans, and Myristica malabarica (Bombay nutmeg).
Toxicity: 3
Method: Injected and swallowed
Symptoms: Anxiety, blurred or double vision, convulsions, dehydration, dry mouth, euphoria, fever, flushing, hallucinations, irregular heartbeat, nausea, pain, restlessness, and tachycardia.
Description: Some information about nutmeg.

Chapter 3

Street Drugs

LSD

Scientific Name: Lysergic acid diethylamide.
Other: Lysergide.
Toxicity: 2
Method: Injected and swallowed
Symptoms: Coma, confusion, convulsions, excitement, hallucinations, psychosis, and spasms.
Description: Some information about LSD that includes a reference to nutmeg.

Chapter 4

Biochemical Warfare

Botulinum

Scientific Name: Clostridium botulinum.
Other: Botox and botulism.
Toxicity: 6
Method: Injected and swallowed
Symptoms: Blurred or double vision, nausea, and paralysis.
Description: Some information about the botulinum toxin used as a bioweapon.

Order by Toxicity

Toxicity Rating 6
Botulinum, 1, 7
Toxicity Rating 4
Ammonia, 1
Toxicity Rating 3
Nutmeg, 3
Toxicity Rating 2
LSD, 5

By Method

Breathed
Ammonia, 1
Swallowed
Nutmeg, 3
Injected
Botulinum, 1, 7
LSD, 5
Nutmeg, 3

Figure 1: Standalone entries: Household Poisons

Figure 2: Standalone entries: Plants

Figure 3: Standalone entries: Street Drugs

Figure 4: Standalone entries: Biochemical

Figure 5: Order by toxicity

Figure 6: Order by method
By Symptoms

Vital Signs
Fever/Hyperthermia
Nutmeg, 3

Tachycardia/Rapid Heartbeat or Pulse
Ammonia, 1
Nutmeg, 3

Head, Eyes, Ears, Nose, Throat
Blindness
Ammonia, 1

Blurred or Double Vision
Botulinum, 1, 7
Nutmeg, 3

Dry Mouth
Nutmeg, 3

Lip/Mouth Irritation
Ammonia, 1

Figure 7: Order by symptom

Figure 8: Index

Index

ammonia, 1
ammonium hydroxide, see ammonia

Bombay nutmeg, see Myristica malabarica
botoc, see botulinum
botulism, see botulinum

Clostridium botulinum, see botulinum

LSD, 5

lysergic acid diethylamide, see LSD
lysergide, see LSD

M
Myristica argentea, see nutmeg
Myristica fragrans, see nutmeg
Myristica malabarica, see nutmeg

N
nutmeg, 3, 5

P
Papuan nutmeg, see Myristica argentea

Nicola L. C. Talbot
Transparent file I/O using the original \TeX\ program and the plain \TeX\ format

Udo Wermuth

Abstract
Research papers demonstrate that it is possible to use a \TeX\ file to distribute malware to a victim’s system. Although it seems that no report has been published about a virus of this kind in a real attack, the potential danger to abuse a \TeX\ source file to transport unfriendly code exists. This article explains an idea to make \TeX\’s file I/O more transparent and develops requirements to turn the idea into \TeX\ macros. Their application in a \TeX\ file received from an untrusted source identifies all file names used for I/O operations. But the macros demand concentrated work with numerous text inputs and a non-beginner’s knowledge of \TeX.\ Furthermore, users should be patient, curious, and courageous.

1 Introduction
The usual input to \TeX\ is a plain text file containing a few control sequences to instruct the program how to format the document. Through its macro capabilities \TeX\ allows an author to increase the number of recognized control sequences, tailoring them to the needs of the text. But \TeX\ does not forbid writing a macro like “\texttt{\texttt{\def\texttt{useless}\	exttt{useless}}}” which generates an endless loop when \texttt{useless} appears in the text. (Such endless loops are inherent for a macro expansion language [9, p. 659].) Similarly, some control sequences implemented directly in the \TeX\ program—these are named primitives—must be used with care. For example, the simple “\texttt{\texttt{\openout0=\jobname\bye}}” truncates the file name to which \jobname expands, plus extension .tex, with zero bytes. As this is usually the file that \TeX\ processes as the main file in the current run its original contents are gone.

Thus it’s easy to waste CPU cycles by executing \texttt{useless}. On a modern multiuser system the single-threaded \TeX\ program occupies at most one CPU and a reasonably configured \TeX\ system doesn’t require much main memory. So other users are hardly affected in their own work unless many \TeX\ programs run \texttt{useless} in parallel. To produce a file that should be loaded by \texttt{\input} in a co-worker’s \TeX\ source file with the above \texttt{\openout} statement is a bad joke and might become a disaster if there is no backup of a laboriously created main file. (To protect yourself in such a case from this bad joke set your main file temporarily to read-only, for example, under Unix-like systems with \texttt{chmod u-w}.)

These examples raise the question: how brave or careful must one be to typeset a \TeX\ file received from a friendly joker, a well-known silly person, an inexperienced beginner, a person known only by name, or an unknown individual who makes files available for downloading on the Internet. Is it possible that the \TeX\ run of this plain text file results in a damaged or, worse, virus-infected system?

Unfortunately the answer is: Be careful! A \TeX\ run using a specific prepared plain text file might delete important files, read private data, or infect your local system with a computer virus.

Published attacks. The thesis [13] uses L\LaTeX\ and GNU Emacs to show in a feasibility study that a plain text file can contain code that spreads itself to other plain text files. In [1, 2] an ε-\TeX\ source includes instructions to create during the compilation a \texttt{\texttt{\script file}} in a certain directory. The execution of this file infects computers running MS Windows—the \TeX\ source contains an absolute path that’s only valid for this operating system (OS).

The attacks are possible as \TeX\ contains commands to read from and write to any file. Some implementations of \TeX\ restrict which directories are permitted for \TeX\’s I/O primitives. Of course, every OS should protect itself and mechanisms are usually in effect for ordinary users. But what can be done if the user runs \TeX\ with system administrator rights? Or when the system administrators of a multiuser system that provides a \TeX\ service configured the system in a way that private information is accessible to users without a need to know [12]?

I found no report of any real attack in which someone was the victim of a \TeX\ source file transporting a virus. This risk seems to be very small. But we can assume that some users have coded an endless loop and a few users have deleted an important file with an inappropriate file name for an \texttt{\openout}.

Is \TeX\ an insecure program? No, definitely not. Both published attacks need supporting tools: the programmable GNU Emacs or a \TeX\ \script\ file placed in an auto-start directory. Similar to an email, \TeX\ source can be abused to transport malicious code. We avoid clicking on a link in an email sent by an unknown person and we must be cautious if we execute a \TeX\ file received from an untrusted source. Sure, \TeX\ could be more verbose with file names. But it doesn’t help to learn which file was deleted and it’s very cumbersome if \TeX\ asks every time for the user’s permission to process a file, as we will see.

It’s somewhat pointless to ask today why \TeX\ wasn’t programmed with a more restricted access to files. I only provide three observations. First,

doi.org/10.47397/tb/43-1/tb133wermuth-trio
at the time \TeX was designed, this program tried to achieve new inconceivable advancements in typesetting. The limits of the available computers were touched; for example, memory had to be conserved. Second, Don Knuth’s intention, when he began the design, was to create a tool for his secretary and himself [9, p. 606; 10, p. 63]. There was no reason for mistrust, i.e., bad jokes were not expected. Third, the original \TeX was reimplemented as \TeX82 and at that time portability was a major concern [8, p. 254]. As file names are highly OS-dependent, \TeX’s code cannot cover all possibilities and must be carefully customized through a change file [8, pp. 123–124].

Implementors often transfer \TeX’s archaic default file system into a nearly unrestricted model for the target OS. But excluding absolute paths or paths containing the short-cut for the parent directory (i.e., “..”) inhibit the attacks of [1, 2]. The recommendation of [7, §511], to use portable file names built only from letters and digits may be too restrictive, yet reminds us to think about simplicity.

Other risks. Modern \TeX implementations, not the original one that is used in this article, activate a communications mechanism to the OS; this feature uses the stream number 18 in `write' statements. That such a communication makes the life easier for viruses and their developers or crackers (to name them in accordance with [15]) has been known for a long time (see [11, p. 454, no. 3]. Thus, the `write18' feature is often disabled by default and must be explicitly switched on by the user.

A cracker might hide the use of a `write18'. Therefore, always distrust tricky code without appropriate comments. For example, a single search for `write18' fails with this obfuscated code; see [14].

```
\lccode'='r\lccode'q='w\lccode'='t\lccode'u='i
\lccode'='e\lccode'a='\expandafter\lccode'o\\e\lccode'='i\lccode'='\csname qeurw\endcsname
\newcount\make\make=9 \multiply\make by2 \immediate\trouble\make\echo === \text{GOTCHA} ===
```  

All computer users know that all operating systems require regular updating to reduce the risk of a cracker getting into a system through security holes. Additional risks exist that stem from the installation of a distribution (see, for example, [16]) or that are given through the tools of the OS which are required to process a \TeX source file and \TeX's output; see section 10. From all I know, these risks are much higher than the danger coming from a plain text file containing \TeX commands.

Unfriendly code can lurk everywhere. Even if you compile carefully inspected source code yourself, malicious code can be present [18].

Protection by inspection. The abovementioned articles about possible attacks need several lines of \TeX code so a look at the source file might reveal the presence of instructions for a virus. But a cracker might try to hide the coded malware. Thus the \TeX files one gets from an unknown or untrusted source must either be executed in a restricted environment or be the subject of a thorough visual inspection.

A journal or proceedings editor receives numerous source files and it’s unlikely that all authors are known by the editor. On the other hand, the authors want to have their articles published and not be accused of spreading bad code. Nevertheless, an author might be a victim and unknowingly send out a \TeX file transporting code for a virus.

Although it’s a significant effort, editors should perform a visual inspection as part of the editorial work. I assume that they review text and code in most cases. Besides security, other reasons make this necessary as not all authors are willing to follow the instructions of the journal; some prefer to cheat. For example, look at the report [4] about problems with the length of submitted papers.

Protection by macros. This article describes a set of macros for the original \TeX engine with the plain \TeX format to make the file I/O operations more transparent. By this I mean that a user controls which files are processed when \TeX executes `input', `openin', or `openout'. The macros don’t detect instructions for a virus or state that a file shouldn’t be processed; they only report which file names occur and give the user a chance to change them. But they accomplish more: The instrumented source file cannot stealthily bypass their reporting.

One goal of the macros is to produce an identical DVI file compared to a run without the macros if the original source is error-free. Section 3 discusses why this goal cannot be reached for all plain \TeX source files; a few eccentric constructions might fail.

Of course, the macros need a few resources. Besides memory space for the macros and other control sequences, the macro package declares five token registers. Thus, one cannot use the macros in the unlikely case that a source file requires more than 238 non-scratch token registers. Sure, the dreadful \TeX “capacity exceeded” error message occurs earlier if the macros are used. But this is merely a theoretical problem as modern \TeX installations set \TeX’s compile-time constants so high that it’s doubtful that an error-free source reaches \TeX’s limits even if the macros of this article are active.

Usefulness of the macros. Above I wrote that the risk to become a victim of a virus that enters a

Udo Wermuth
system via a plain \TeX file is very small. Nevertheless it might be an interesting intellectual pastime to see how to protect a system with macros against malicious code. Moreover, such macros may reassure people and increase confidence in \TeX’s security.

A cracker might be aware that these macros exist and avoid conspicuous actions if they are present. Or, say, the code contains a test so that it gets executed only on Sundays and thus a check that runs on a Thursday doesn’t detect it. Clearly the macros cannot help to protect a system if they are not active during all executions of a source file.

Although I think a cracker cannot circumvent the macros if the user follows all usage instructions carefully, everyone uses the macros at one’s own risk.

2 Primitives requiring file names

With the procedure \texttt{scan\_file\_name} \cite{537}, \TeX scans in a system-independent way file names. Although file names are highly system dependent, this aspect is handled in other sections of the program. Here I use the convention that a file name consists of an optional path, the main part of the file name, and an optional extension. The path is a sequence of directories with a slash after each directory name; a period separates main part and extension. Spaces are forbidden in file names. A single period in the path, i.e., \texttt{.}, stands for the current directory, and \texttt{../} represents the parent directory.

The above-mentioned procedure is used in the implementation of four primitives: \texttt{\input} in \S \texttt{537}, \texttt{\font} in \S \texttt{1257}, \texttt{\openin} in \S \texttt{1275}, and \texttt{\openout} in \S \texttt{1351}.

The primitive \texttt{\font} is somewhat special in this list. \TeX expects a file name but replaces any extension with \texttt{tfm} \cite{562} as it reads for \texttt{\font} only files containing \TeX font metric (TFM) data. It checks that the contents of the file with the constructed name obey the specifications of TFM files \cite{562}.

Although this sounds simple it might be very hard to determine which font \TeX loads. Above it was shown that the flexibility of \TeX can be abused to hide what the code will do. File names are no exception, as the following input proves.

\begin{verbatim}
\def\gobble#1{}\lccode'z='z \show\-input hello \expandafter\expandafter\expandafter\input\input hello
\end{verbatim}

What does \texttt{\show\next} in the last line display?

I don’t see any way to abuse the primitive \texttt{\font} to read a file that isn’t a TFM file.

Three main primitives. The primitives \texttt{\input}, \texttt{\openin}, and \texttt{\openout} use the complete file name that they receive. They append the extension \texttt{.tex} if \TeX doesn’t find one \cite[pp.~25, 217, 226]{206}. With \texttt{\input} and \texttt{\openout}, \TeX prompts for a new file name if the file cannot be found or opened for writing, respectively \cite[§§530, 537, 1374]{537}. The primitive \texttt{\openin} never asks the user to enter a new file name \cite[p.~325, no.~582 of \TeX’s error log]{562}. When \TeX asks for another file name, the good news is that it displays first \texttt{! I can’t find file} or \texttt{! I can’t write on file} followed by the file name that it had scanned. Thus, even if the file name was entered in an obfuscated manner now the user sees the name.

3 Expected problems

Primitives and macros behave differently in a \TeX run. If the three file I/O primitives are replaced by macros, under what circumstances does this influence the typesetting? Sure, a source file might test these command names and produce a different DVI file if one of them is a macro. In this case I only care about the result obtained with file I/O macros.

One important difference lies in the ability of macros to expand. The primitives \texttt{\openout} and \texttt{\openin} are allowed in an \texttt{\edef} (or \texttt{\def}, \texttt{\write}, etc.) so the macros should be accepted too. Thus the macros must either contain only expandable tokens and be quite simple or stop the expansion early.

The primitive \texttt{\input} is a special case as its acceptance in an \texttt{\edef} depends on the contents of the file that is input. \TeX usually throws an error, as it treats the end of a file that’s input similar to an outer macro \cite[p.~206]{206}. But \TeX accepts a file that ends with the primitive \texttt{\noexpand}. Thus, the macro \texttt{\input} must be completely expanded and do its work. But if this macro, say, sets a Boolean flag from false to true, \TeX runs into an error if \texttt{\input} is executed in an \texttt{\edef}. This is completely independent of the contents of the file that gets input.

This is expected, as \texttt{\input}s expansion is null but \TeX starts to read from the file \cite[p.~214]{206}. Thus, use of \texttt{\expandafter} will also give different results. For example, \texttt{\expandafter\show\input hello} displays “the letter H” if the file \texttt{hello.tex} contains the text “Hello \TeX!” But a macro for \texttt{\input} expands just one level and \TeX displays its first token, i.e., \texttt{\show} inactivates this token. (Our macro will start with \texttt{\begingroup}; so any control sequence between \texttt{\expandafter} and \texttt{\input} that reads at least one argument and doesn’t open an unclosed

Transparent file I/O using the original \TeX program and the plain \TeX format
group gives an error.) Similar problems exist with the macros for \openin and \openout.

This “contents dependency” for the acceptance of the primitive \input makes it possible to place it between `\csname` and `\endcode`. TEx allows this if the file that’s input expands to character tokens only; \openin and \openout are always rejected. For example, the statement `\csname\input hello \endcode` is a valid construction. Usually a macro fails in this scenario if it isn’t very simple.

A similar situation occurs with the application of a prefix, `\number`, etc., to the primitive \input. The first token of the file that’s input must accept this command or \TeX displays an error; \openin and \openout don’t accept such commands.

A reader might agree with me in finding some of these constructions weird and classify them as bad programming practice. Nevertheless the macros will address the four problems: the `\csname` problem”, the “\edef problem”, the “\expandafter problem”, and the “apply problem”. Some can be solved interactively, others require a change of the source. The important point is: Be alert if a source file uses one of these unusual constructions and check the code carefully to convince yourself that it is required.

Note: The discussion concentrates on plain \TeX but, for example, TUGboat uses its own macro package in which the command `\input` becomes a macro. Now, \TeX always throws errors for the `\edef` and `\csname` problems but not for `\global` as the macro absorbs it; `\long`, `\number`, etc., give errors. Macros with at least three arguments in the `\expandafter` problem hinder `\input`.

Privacy. Let’s state it frankly: It’s not possible to hide the fact that file I/O primitives are replaced by macros. This doesn’t mean that all macros must be made public but it means that I decided not to change, for example, `\meaning`, so a cracker can look at the macro `\input`. Thus, a cracker knows which control word was given the original meaning of the primitive as it is called in the macro.

The important question is, what can a cracker do with this information? It’s suspicious to input a file without using the macro. A user sees on the terminal that \TeX inputs a file except if `\batchmode` is active. My advice: Stop the execution if this happens without the approval through the procedure of the macro described in section 4. Thus the first statements of the macro package are

\begin{verbatim}
\let\batchmode=\scrollmode
\let\nonstopmode=\scrollmode
\end{verbatim}

to make sure that no file can be input without a message on the terminal.

I deactivate `\nonstopmode` too in order to assure that \TeX stops if it cannot find a file as I decided to let `\input` scan all file names with a trick that makes \TeX prompt for a new file name. Then the user has the chance to check which file gets processed and to change the file name if necessary or to end the run. In a second step the file name is given to the primitive whose name occurs in the source to process the file, if the run wasn’t canceled.

Another source file might redefine the primitives used in our macros and then they might not do what is intended. This problem gets solved in the usual manner: The used primitives are copied to new control words with a unique start sequence. I use the string “TRIO” for these copies and “TrIO” for all private macros. For example, instead of the primitive `\begingroup` I use `\TRIObegingroup`. The source might use the prefix TRIO too, for example,

\begin{verbatim}
\def\TRIObegingroup{% open three groups
  \begingroup \begingroup \begingroup}
\end{verbatim}

(how likely is this?) and our own macro must get a new name, for example, `\TRIxObegingroup`.

Security. The primitives `\openin` and `\openout` are not as verbose as `\input`. They operate on a file without stating the file name on the terminal (or in the log file). The control words that save the meaning of these primitives must not be made public. Otherwise an evil-doer circumvents the macros and applies the original primitives under their new name.

Fortunately, none of our public macros require the control words with the original meaning of these two primitives as `\input` is executed first. As mentioned above the file name is read with a trick to make \TeX ask for a new file name. The user must enter a special file name that in a next step contains control words that have received via `\let` the meaning of either `\openin` or `\openout`. Therefore these control words can be given what I call a password-protected name.

A password-protected name contains a string of at least six letters in upper- and lowercase and with one letter from the first third of the alphabet and another from the last third. If the six letters form neither an English word nor a word in the language of the user it is very unlikely that this control word can be guessed or computed by a cracker. (Six letters define the minimum; use more if you like. Shorter passwords might be discovered with \TeX through a brute force attack.) For example, I use in this text the name `\TRIOaAmNzZopenin` in a `\let` assignment to save the meaning of the primitive `\openin`. Note, “aAmNzZ” is a placeholder that must be changed.

Udo Wermuth
by the user if the macros are used. First, it’s the
default that a cracker knows; second, it’s much too
too simple to make a good password.

The macros contain several passwords and some are
applied more than once. For example, every used \TeX
primitive has not only a copy with the prefix
“TRIO” but also one with the prefix “TRIO\hJqsS”
built with the password “h\JqsS” — again this is a
placeholder which must be changed before the macro
package is used. During the run a check procedure
gets occasionally called to assure that both control
words have the same meaning. At the start we define

\let\TrIOhHJqsSfifx\TRIOifix
\let\TrIOhHJqsSelsefxfi\TRIOifx
\let\TrIOhHJqsSifxfi\TRIOelse
\let\TRIOhHJqsSfi\TRIOifx % \TRIOfi is OK
\let\TRIOhHJqsSelse\TRIOhHJqsSstop{TRIOelse} % \TRIOelse is OK
\let\TRIOhHJqsSif\TRIOhHJqsSfi % \TRIOifx is OK
\let\TRIOhHJqsSf\TRIOhHJqsSf % \TRIOf is OK
... \% many more \let assignments
\def\TrIOhHJqsSstop#1{\TrIOhHJqsSerrmessage{TrIO
ALERT !!! Don’t trust the source (#1)}}
\def\TrIOoddDJqWcheck{% check that macros are OK
\TrIOhHJqsSif\TrIOhHJqsSelse\TrIOhHJqsSstop{TRIOifix} %
\TrIOhHJqsSf % \TRIOifix is OK
\TrIOifix\TrIOhHJqsSelse\TrIOhHJqsSelse % \TRIOelse is OK
\TrIOhHJqsSf % \TrIOhHJqsSelse is OK
\TrIOelse\TrIOhHJqsSstop{TRIOf} %
\TrIOhHJqsSf % \TRIOhHJqsSfi is OK
... \% many more \ifx tests

An undetectable problem. As mentioned above the
macros for \texttt{openin} and \texttt{openout} input a special
file. Changes in the category codes (or catcodes)
of used characters might change what the file shall
accomplish. Thus, I decided to reset all letters and
some symbols to their default catcodes before the
macros of the special file are executed. This — as
well as other decisions like the use of \texttt{\count255} —
requires executing the code of the macros most of the
time inside a group. Sure, \texttt{input} should not load
the file inside a group. But \texttt{openin} and \texttt{openout}
act globally and can be placed inside a group.

In order to keep such changes local to the group
they must not be prefixed by \texttt{global}. The problem
occurs if the source sets \texttt{\globaldefs=1} because
then every assignment, prefixed by \texttt{global} or not,
becomes global. Code like this is ok:

\begingroup \input hello \endgroup

Our macro \texttt{\input} sets \texttt{\globaldefs=0}, executes
its code, and sets \texttt{\globaldefs=1}. The first assignment
to \texttt{\globaldefs} inside the macro, i.e., \texttt{\globaldefs=1},
is always global. Thus a problem occurs if
\texttt{\globaldefs} was set to \texttt{-1} before the above group
as then \texttt{\globaldefs} is restored as \texttt{0} rather than \texttt{-1}
after \texttt{\endgroup}. Similarly the code \texttt{\globaldefs=1}
\begingroup \input hello \endgroup restores \texttt{0}
not \texttt{1} for \texttt{\globaldefs} after \texttt{\endgroup}.

A positive \texttt{\globaldefs} is rare, and when it
does occur it is usually in the good case above. But
the problem that arises from the two bad cases can
be neither solved nor detected. The macros can only
report that \texttt{\globaldefs} is positive. The user must
then carefully check the source to understand why
this seldom-used integer parameter was set.

4 The macro \texttt{\input}

Do we need to make \texttt{\input} more transparent, as
it writes the received file name to the terminal if
\texttt{\batchmode} is inactive? It’s easy to miss one file in
a flood of output on the terminal. I prefer to check
which files are input and I want to have the control
to redirect the request. It is crucial for success to
check which files are input. For example, a user must
never allow that a source inputs any of the files of
the macro package and continues the run.

The trick. How does the macro force \TeX to ask
for a new file name? A nonexistent path is placed
in front of the given file name. For example, I define
\texttt{\input \TrIOnosubdir\nosubdir/} where \texttt{\nosubdir/}
must not exist as a directory in the current
current directory. Next, the macro changes
\texttt{\input hello} into
\texttt{\textbackslash input \TrIOnosubdir hello}.

This works fine as long as the file name doesn’t
start with “../” as this might undo in some \TeX
implementations the “nosubdir/” and the remaining
path points to a file in the current directory that
contains the same name as the file that should be found
in the parent directory. In such a case an existing
file is input without asking the user. The user sees
on the terminal that \TeX inputs a file without ap-


\let\TrIOhHJqsSfifx\TRIOifix
\let\TrIOhHJqsSelsefxfi\TRIOifx
\let\TrIOhHJqsSifxfi\TRIOelse
\let\TRIOhHJqsSfi\TRIOifx % \TRIOfi is OK
\let\TRIOhHJqsSelse\TRIOhHJqsSstop{TRIOelse} % \TRIOelse is OK
\let\TRIOhHJqsSif\TRIOhHJqsSfi % \TRIOifx is OK
\let\TRIOhHJqsSf\TRIOhHJqsSf % \TRIOf is OK
... \% many more \ifx tests

An undetectable problem. As mentioned above the
macros for \texttt{\input} input a special
file. Changes in the category codes (or catcodes)
of used characters might change what the file shall
accomplish. Thus, I decided to reset all letters and
some symbols to their default catcodes before the
macros of the special file are executed. This — as
well as other decisions like the use of \texttt{\count255} —
requires executing the code of the macros most of the
time inside a group. Sure, \texttt{\input} should not load
the file inside a group. But \texttt{\openin} and \texttt{\openout}
act globally and can be placed inside a group.

In order to keep such changes local to the group
they must not be prefixed by \texttt{global}. The problem
occurs if the source sets \texttt{\globaldefs=1} because
then every assignment, prefixed by \texttt{global} or not,
becomes global. Code like this is ok:

\begingroup \input hello \endgroup

Our macro \texttt{\input} sets \texttt{\globaldefs=0}, executes
its code, and sets \texttt{\globaldefs=1}. The first assignment
to \texttt{\globaldefs} inside the macro, i.e., \texttt{\globaldefs=1},
is always global. Thus a problem occurs if
\texttt{\globaldefs} was set to \texttt{-1} before the above group
as then \texttt{\globaldefs} is restored as \texttt{0} rather than \texttt{-1}
after \texttt{\endgroup}. Similarly the code \texttt{\globaldefs=1}
\begingroup \input hello \endgroup restores \texttt{0}
not \texttt{1} for \texttt{\globaldefs} after \texttt{\endgroup}.

A positive \texttt{\globaldefs} is rare, and when it
does occur it is usually in the good case above. But
the problem that arises from the two bad cases can
be neither solved nor detected. The macros can only
report that \texttt{\globaldefs} is positive. The user must
then carefully check the source to understand why
this seldom-used integer parameter was set.

4 The macro \texttt{\input}

Do we need to make \texttt{\input} more transparent, as
it writes the received file name to the terminal if
\texttt{\batchmode} is inactive? It’s easy to miss one file in
a flood of output on the terminal. I prefer to check
which files are input and I want to have the control
to redirect the request. It is crucial for success to
check which files are input. For example, a user must
never allow that a source inputs any of the files of
the macro package and continues the run.

The trick. How does the macro force \TeX to ask
for a new file name? A nonexistent path is placed
in front of the given file name. For example, I define
\texttt{\input \TrIOnosubdir\nosubdir/} where \texttt{\nosubdir/}
must not exist as a directory in the current
current directory. Next, the macro changes
\texttt{\input hello} into
\texttt{\textbackslash input \TrIOnosubdir hello}.

This works fine as long as the file name doesn’t
start with “../” as this might undo in some \TeX
implementations the “nosubdir/” and the remaining
path points to a file in the current directory that
contains the same name as the file that should be found
in the parent directory. In such a case an existing
file is input without asking the user. The user sees
on the terminal that \TeX inputs a file without ap-

Transparent file I/O using the original \TeX program and the plain \TeX format
The first line (\TRIObegin{group}) makes \TeX stop if the “apply problem” occurs or if the macro is expanded inside a \cocode{\csname \cname \endcsname} structure. Line 2 switches off the application of a token held by the primitive \afterassignment; see section 5. The tricky code works in an \edef too; see below. The definition of an undefined control word catches the expansion of \input in an \edef. The macro \TRIOhandleglobaldefs handles the \globaldefs problem described in the previous section. All these technical parts are discussed in a moment.

The important parts: \TRIOcountiocmd, catcode changes in \TRIOsetcatcodes, the \TRIOinput file, \TRIOinputmessage, and the last line’s trick.

The first macro counts the number of times one of the three file I/O primitives is called.

\begin{verbatim}
def\TRIOcount{0 }\countdef\TRIOcount=255
def\TRIOcountiocmd{% increment \TRIOcnt}
def\TRIOinputmessage{% establish a few \catcodes\cocode{\csname \input \endcsname} structure. \endcsname}
def\TRIOglobaldefs{% inform about \globaldefs if necessary after it closes the group opened in the first line of \input.}
def\TRIOhandleglobaldefs{% inform about \globaldefs if necessary after it closes the group opened in the first line of \input.}
\end{verbatim}

Together with information written to the terminal and the log file a simplified procedure for repeated execution of the source can be realized; see section 9.

\begin{verbatim}
def\TRIOglobaldefs{0 }% inform about \globaldefs : if \globaldefs>0 and switch to \globaldefs=0
\end{verbatim}

The variant is called if the source file contains \TRIOmacros. The user must enter this macro into the source to fix some of the discussed problems. This file is input in \TRIOinput and the log file a simplified procedure for repeated execution of the source can be realized; see section 9.

\begin{verbatim}
def\TRIOglobaldefs{1 }% inform about \globaldefs : if \globaldefs>0 and switch to \globaldefs=0
\end{verbatim}

\begin{verbatim}
\TRIOglobaldefs=0 \TRIOglobaldefs=1 \TRIOglobaldefs=0 \TRIOglobaldefs=1 \TRIOglobaldefs=1 \TRIOglobaldefs=1 % \TRIOelse \TRIOsuspendafterassignment \TRIOglobaldefs=0 \TRIOglobaldefs=1 \TRIOglobaldefs=0 \TRIOglobaldefs=1 \TRIOglobaldefs=0 \TRIOglobaldefs=1 \TRIOglobaldefs=0 \TRIOglobaldefs=1 \TRIOglobaldefs=0 \TRIOglobaldefs=1 % \TRIOifnum\TRIOglobaldefs>0 \TRIOmessage
% \globaldefs>0 and switch to \globaldefs=0
\TRIOglobaldefs=0 \TRIOmessage
\end{verbatim}

A variant. To address some of the problems discussed in the previous section a second macro for \input is coded. It carries a password-protected name, \TRIOCcPzinput, to avoid its unnoticed use. It differs from the macro shown in two respects:

1. The message states “INPUT” instead of “input” to identify itself to the user.
2. In front of \TRIOinput in the last line the macro \TRIOCcPzXtransfer appears.

The variant is called if the source file contains \TRIOCcPzXmove. The user must enter this macro into the source to fix some of the discussed problems.

\begin{verbatim}
def\TRIOCcPzXmove\input{% transfer tokens
\end{verbatim}

Use this macro only if you are convinced that a \csname, \expandafter, or “prefix” is required and the source cannot extract the password in the name.

An example. Most macros of this article are bundled in the file \TRIOmacros. This file is input in the first line of the source file that should be checked.

\begin{verbatim}
\TRIOglobaldefs=0 \TRIOglobaldefs=1 % inform about \globaldefs : if \globaldefs>0 and switch to \globaldefs=0
\TRIOglobaldefs=1 % inform about \globaldefs : if \globaldefs>0 and switch to \globaldefs=0
\end{verbatim}

When this file is executed \TeX displays the messages of the macro followed by an error:

\begin{verbatim}
... \cocode{\csone} \input hello \errorstopmode
\end{verbatim}

\begin{verbatim}
... \cocode{\csone} \input hello \errorstopmode
\end{verbatim}

These catcodes are fixed and build the base for the catcode changes in the file \TRIOinput:

\begin{verbatim}
\cocode{\csname \input \endcsname}
\end{verbatim}

Handling \globaldefs. The macro that checks the setting of \globaldefs clears it if it is positive as explained earlier. This macro de- and reactivates \afterassignment in case it holds a token: The macro \TRIOsuspendafterassignment blocks the application of this token after an assignment and the macro \TRIOinitafterassignment restores the default behavior. Finally, the macro defines the macro \TRIOendgroup that resets the integer parameter \globaldefs if necessary after it closes the group opened in the first line of \input.
ports that the first I/O command was found in line 2 and that this command is \input, and finally the macro displays what to do next.

We enter “hello” as the new file name. \TeX shows in the next line that it inputs hello.tex. But then an error message pops up.

(hello.tex)
! Missing \endcsname inserted.
<to be read again>
\TRIObegingroup
\input ->\TRIObegingroup
\TRIOafterassignment... 
1.3 \csname \input hello \endcsname
?
This error message signals the \csname/\endcsname problem. The answer to the question mark is to type “42”, then to insert the correct code, i.e., \texttt{\csname}, at the next prompt. Finally, enter the file name.

? 42
\input ...\TRIOendgroup
\TRIOinput \TRIOnosubdir
1.3 \csname \input hello \endcsname
?
! I\csname
I can't find file ‘nosubdir/hello.tex’.
1.3 \csname \input hello \endcsname
\endcsname
Please type another input file name: hello
(hello.tex)

In this example the \csname problem was fixed successfully. But, for example, the code \texttt{\csname AA \input hello \endcsname} would create a different typesetting result compared to the original source. Check carefully if the macro \texttt{\TRIOcCkPxXmove} can be inserted, if the contents of the file can be typed in, or if the source file should be rejected.

The next line represents such an insertion by the user. Now the “normal” case occurs except that the word “INPUT” signals the use of the macro. (Note the “2” as the “42” skipped the counting.)

<<<
(TrIOinput.tex)
TrIO >>> ( 2 ) Line 4: INPUT
>>>
enter shown file name without ‘nosubdir’/
 <<<
! I can't find file ‘nosubdir/hello.tex’.
1.4 \TRIOcCkPxXmove\global\input hello

Please type another input file name: hello
(hello.tex)
without the macro \TeX reports “! You can’t use a prefix with ‘\begingroup’,” and the fix is to enter “42” and “\texttt{\TRIOglobal}”. The apply problem can always be solved in this way.

Next, an error message appears as the contents of hello.tex doesn’t start with an assignment; it’s an error in the original source: “! You can’t use a prefix with ‘the letter H’.”

After pressing RETURN \TeX displays

> \TRIObegingroup=\begingroup.
\input =\TRIObegingroup
\TRIOafterassignment...
1.5 \expandafter\show\input
hello

which is not an error message but the result of the primitive \show. Nevertheless the macro \input lost its first token. Without intervention \TeX will display an error message as soon as it reads the corresponding \endgroup. This time the interactive fix is to type “41” followed by “\expandafter\show”.

With a macro that reads arguments instead of the non-typesetting command \show such a fix is not possible. Edit the source and use \texttt{\TRIOcCkPxXmove} except in cases like \expandafter{\input hello }, in which the \expandafter should be deleted.

The macros in \texttt{TrIOMacros.tex} are designed in a way that all errors in the original source produce errors in the instrumented file, although the error messages and/or recovery might be different. An erroneous source might lead to an instrumented source in which it is impossible to recover from an error during the execution.

The last line in the above source gives an example of such an error. In the original source \TeX displays “Runaway definition?” but the instrumented source shows first “! Undefined control sequence.”

\texttt{\TRIOfixedef
1.6 \edef\csone{\input
hello }\show\csone\bye
}

The name of the undefined control sequence informs the user what to do: Skip 36 tokens and insert then \TRIOfixedef. Doing so and after entering “hello” \TeX displays the original error message.

? 36
\input ...\TRIOendgroup
<<<
(TrIOinput.tex)
TrIO >>> ( 2 ) Line 4: INPUT
>>>
enter shown file name without ‘nosubdir’/
 <<<
! I can't find file ‘nosubdir/hello.tex’.
1.4 \TRIOcCkPxXmove\global\input hello

Please type another input file name: hello
(hello.tex)

Runaway definition?

->\TRIObegingroup \TRIOafterassignment \ETC.

! File ended while scanning definition of \csone

Transparent file I/O using the original \TeX program and the plain \TeX format
Next \TeX{} complains about too many closing curly braces as in the original source.

The \show\csone displays:
\begin{verbatim}
\TRIObegingroup \TRIOafterassignment \TRIOempty \TRIOdef \TRIOempty {} \TRIOendgroup Hello TeX!
\end{verbatim}

so that \csone contains more material than in the source file; a prefix or \texttt{number}, etc., now gives a new error if the original accepts this in front of \csone.

Summary: A user can fix the problem interactively, but not always the \texttt{name} and the \texttt{expandafter} problems; one can try to fix them in the source. The \texttt{def} problem must be fixed interactively but the defined macro has additional tokens.

5 Macro for \texttt{afterassignment}

Next, let’s look at the support macros that we need to handle the primitive \texttt{afterassignment}. This primitive stores a single token that isn’t expanded \cite[p. 215]{6}; thus it can hold an undefined macro and execute it after it was defined. To reproduce this behavior the macro must store the token in a token register and not via a \texttt{let} assignment. On the other hand, a curly brace cannot be placed in a token register; this requires \texttt{let}. To distinguish these cases the macro sets a flag. \texttt{afterassignment} cannot appear in a \texttt{name} \texttt{endsname} construction or with a prefix like \texttt{global}.

\begin{verbatim}
\newif\ifTRIOisupdatedtoken % true: token is stored
\newif\ifTRIOafterassignment % true: don’t % insert a token after an assignment
\newif\ifTRIOusetokenlist % true: use token reg % the token register
\end{verbatim}

A second difficulty is that \texttt{afterassignment} can be used in an \texttt{edef} or \texttt{def} but the macro would fail if it is fully expanded. Therefore a second token register is declared to stop the expansion.

\begin{verbatim}
\let\TRIOafterassignment=\afterassignment\newtoks\TRIOtokendef\TRIOempty\afterassignment % the replacement of the primitive
\def\afterassignment{% \edef expands one level \the\TRIOafterassignment}
\TRIOafterassignment={} \TRIOafterassignment
\end{verbatim}

For the rest of the article — and already in the code just above — I omit the initial \texttt{TRIO} if a primitive is meant and no macro replaces it. For example, above I wrote \texttt{the} instead of \texttt{TRIOthe}, but I will still write \texttt{TRIOinput} since the \texttt{input} primitive has been replaced by a macro.

The main macro blocks the usual work of the primitive \texttt{afterassignment} and then fetches via \texttt{futurelet} the token that should be stored. Two of the other three user macros were shown earlier. One sets the flag to block \texttt{afterassignment}, the second removes this block. The third uses the original primitive to call our own insertion macro.

\begin{verbatim}
\def\TRIOafterassignment{% first save a token \begingroup \endgroup % stop \global
\TRIOglobaldefs \TRIOsavedtokentrue
\futurelet\TRIOsavedtoken=\TRIOchecktoken
% user commands for those who know the macros
\def\TRIOsuspendafterassignment{% switch off \TRIOblockafterassignmenttrue}
\def\TRIOresumeafterassignment{% switch on \TRIOblockafterassignmentfalse % remove block \TRIOinitafterassignment}
\def\TRIOinitafterassignment{% init exec macro \TRIOafterassignment \TRIOAFTERASSIGNMENT}
\end{verbatim}

Again \texttt{globaldefs} must be checked. This is similar to the procedure used for \texttt{input} but here no group must be closed so \texttt{TRIOresetglobaldefs} is defined. It’s called when a token must be stored.

\begin{verbatim}
\def\TRIOglobaldefs{% inform about \texttt{globaldefs}>0 % and switch to \texttt{globaldefs}=0 for the macros
\ifnum\globaldefs>0 \TrIOmessage{TrIO Info: \texttt{globaldefs} is >0 (store)}% \TRIOafterassignment \TRIOresetglobaldefs \TRIOafterassignment \TRIOinitafterassignment \globaldefs=0 \def\TRIOresetglobaldefs{% \TRIOblockafterassignmentfalse \TRIOafterassignment \TRIOinitafterassignment \globaldefs=-1 }%
\else\ifnum\globaldefs<0 % no group, do a reset \TRIOafterassignment \TRIOresetglobaldefs \globaldefs=0 \def\TRIOsetglobaldefs{% \TRIOblockafterassignmentfalse \TRIOafterassignment \TRIOinitafterassignment \globaldefs=1 }%
\else\ifnum\globaldefs<0 % no group, do a reset \TRIOafterassignment \TRIOresetglobaldefs \globaldefs=0 \def\TRIOsetglobaldefs{% \TRIOblockafterassignmentfalse \TRIOafterassignment \TRIOinitafterassignment \globaldefs=-1 }%
\else\TRIOsuspendafterassignment % switch off \TRIOafterassignment \TRIOsetglobaldefs{% and switch on again \TRIOresumeafterassignment}%
\fi\fi\}
\end{verbatim}

The next macro determines the type of the token and stores it either in a token register or via a \texttt{let} assignment.

\begin{verbatim}
\def\TRIOchecktoken{% check token, store a macro \texttt{checktoken} \texttt{save} \texttt{input}
\let\TRIOnext=\TRIOsaveassignedtoken\relax
\let\TRIOnext=\TRIOstoreassignedtoken \TRIOchecktoken % a macro \else % otherwise remove token from the input \let\TRIOnext=\TRIOremovesavedtoken \fi \TRIOnext
\end{verbatim}

\begin{verbatim}
\def\TRIOstoreassignedtoken#1{% \#1: cs in token reg \let\TRIOnext=\undefined \TRIOsettokenlisttrue \TRIOtokendef{(#1)} \TRIOsetglobaldefs}
\end{verbatim}

Udo Wermuth
The application macros just test the flags.
\begin{trioafterassignment}
\ifTrIOusetokenlist % true: use token reg
\else % otherwise use the saved token
\ifTrIOsavedtoken \ifnum\globaldefs>0
\expandafter\expandafter % get rid of
\expandafter\expandafter % the 2 \fi
\expandafter\expandafter % get rid of
\expandafter\expandafter % the 2 \fi
\else \TrIOsavedtokenfalse \fi
\TrIOsavedtokenfalse \globaldefs=1 % & reset
\fi}
\expandafter\expandafter % get rid of
\expandafter\expandafter % the 2 \fi
\\def\TrIOoutputtoken{% output token (check type)
\ifTrIOsavedtoken \ifnum\globaldefs>0
\expandafter\TrIOsavedtoken
\expandafter\fi % with 3+1 \expandafter
\else \TrIOsavedtokenfalse \fi
\TrIOsavedtokenfalse \globaldefs=0 % clear it
\fi

\section{Macro for \texttt{openin}}

Let’s repeat what we already know about \texttt{openin}. It’s nicer than \texttt{input} as it can’t occur in a \texttt{csname/endsname} construction. Moreover, it can’t be prefixed by \texttt{global} as the equals sign here does not mean an assignment is performed; it’s an association between a stream number and a file name. This association acts globally so that we can execute \texttt{openin} inside a group. To solve the \texttt{expandafter} problem in the source just delete this token. But \texttt{openin} might be part of an \texttt{edef}. Thus, the technique of the previous section is applied for \texttt{openin} too.

But \texttt{openin} is also much more unpleasant than \texttt{input}. It operates without stating the file name on the terminal or in the \texttt{log} file. Thus, the control word that saves the meaning of the primitive must not be made public. Otherwise an evil-doer could circumvent the macro and apply the original primitive under its new name. Therefore the copy of the primitive is assigned a password-protected name: \texttt{\texttt{TrIOaAmNzZopenin}}.

The macro \texttt{openin} first reads the stream number, next a test is made to see if the optional equals sign follows, and third \texttt{\texttt{TrIOinput}} is called with the trick so that \TeX asks for a new file name. But this time the user enters two file names. First, a generic file name — for \texttt{openin} it’s by default \texttt{openin} — and then the file name that should be processed by the primitive \texttt{openin}. The file \texttt{openin.tex} contains several password-protected macros that do the important work. Please remember: A user must never allow \texttt{TriOmmacros.tex} or any other file of this package, such as \texttt{openin.tex}, to be processed by the original source.

All aspects of the following macros are either well-known or have been described in the \texttt{openin.tex}, private information is used: a kind of signature that it is the user’s \texttt{openin.tex} and not one by a cracker. A user should change the text to make it unique for each installation. But of course, use only characters whose category codes are known, i.e., set in the list. As \texttt{\texttt{TrIOnext}} becomes undefined in the macro \texttt{\texttt{TrIOaAmNzZopenin}} the message stays private.

\section{Macros for \texttt{openout} and \texttt{immediate}}

The macros to replace the primitive \texttt{openout} are very similar to the ones used for \texttt{openin}; and the Transparent file I/O using the original \TeX program and the plain \TeX format
file openout.tex is similar to openin.tex. The only aspect not yet discussed is the “prefix” \immediate.
\TeX\ allows an \immediate\ everywhere without raising an error for the next token. This is in contrast to, for example, the prefix \long that, after expansion of the next token, requires a definition primitive (\def, etc.) or another prefix (\global, etc.). Although \immediate\ never complains, it influences the next token after expansion only if it is one of \openout, \write, or \closeout.

The way \immediate\ operates means that the macro that replaces the primitive cannot simply set a flag that signals that it was seen. For example, the sequence “\immediate\begingroup\openout” would then faultily apply \immediate\ to \openout. Can we just test if the macro \openout\ follows the macro \immediate? I decided to put an identification primitive at the start of \openout\ so that \TrIOopenout\ doesn’t start with \begingroup but with the sequence “\TrIOimmediate\begingroup”.

The macros for \immediate. As indicated above, the first part of the macros uses the known structure. Only the last line of \TrIOimmediate\ contains a new technique (or trick).

\newif\ifTrIOimoo % true: \immediate\openout \newtoks\TrIOtrimmediate % token reg. for \edef \def\immediate{%\the\TrIOtrimmediate}\% one level \TrIOtrimmediate=%\TrIOimmediate\% expansion \def\TrIOimmediate{% expand the following token \begingroup \TrIOhandleglobaldefs \TrIOimmediate\begingroup \TrIOcount='x\% the trick; explanation follows \endgroup}

\TeX\ treats the alphabetic constant ‘x like a number and digests a space after such a number [7, §442]. To check if a space follows, tokens are expanded (§443) but \TeX\ doesn’t add anything to the alphabetic constant. Thus \TeX\ assigns the value 120 to \TrIOcount after it determines the first token of the expansion of the token that follows \immediate. Only if this first token is \TrIOimmediate\ does the source contain \openout\ as an interim next token for \immediate\ during the expansion.

\def\TrIOimmediate#1{% #1: a token; it’s tested \ifx#1\TrIOimmediate \tripletrue \global\TrIOimottrue \% follows; set flag \else \global\TrIOimofalse \fi \TrIODendgroup \TrIOimmediate\% apply the primitive

A cracker might set the flag (either directly or via \TrIOimmediate\ as the names aren’t protected) to confuse the user. The next \openout\ will use the flag even if no \immediate\ precedes it. Stop the execution if \TrIOmacros\ reports “immediate openout” but the source file seems to have no \immediate\ in front of \openout. Then check the source carefully.

The macros for \openout. As written above, the macros for \openout\ are so similar that they aren’t shown here in detail. Besides the wording “openout” instead of “openin” and “created” instead of “read” in the messages there are two differences:

1. \TrIOopenout\ starts with \TrIOimmediate;
2. the first message in \TrIOOPENOUT\ contains now “\if\TrIOimoo\immediate\fi” in front of the string “openout”.

A new password-protected macro is called in openout.tex; it makes use of the new flag. Otherwise openout.tex is identical to openin.tex.

\def\TrIOBBloYopenout{% get file name from user \read16 to \FilenameOPENOUT\ \if\TrIOimoo\global\TrIOimofalse \let\TrIOnxt=\undefined\fi \TrIOnext\TrIObBlOyYopenout \TrIOcount\FilenameOPENOUT=\undefined \let\TrIOnext=\undefined \let\TrIOnxt=\undefined \TrIOendgroup}

8 The virus example

The following instructions are a modified version of the code containing the virus shown in [1] and [2]. This badly formatted, comment-free but obfuscated code should alert everyone who sees it. (I changed the original source so that it can be executed under plain \TeX. Moreover, the original file names and in one case the contents of a file were changed.)

1. \input\TrIOmacros\ % new 1st line; see below
2. \newif\ifcontinue \continuethrue
3. \def\uncat\{\def#1{c'\#1=12}\dospecials
4. \do\"M\do\ Kasnice\endlinechar='\M
5. \uncat\\readline#1to#2\begingroup\nice
6. \global\readline#1to#2\endgroup
7. \{\newwrite\w\let\c\catcode\c'*13\def
8. *{\afterassignment\d\count255"}\def\d{%
9. \expandafter\c\the\count255=12}{*0D\def\%
10. \a#1"M\immediate\write\w\a\{1\}\c'\M5
11. \newread\r\openin\r=\jobname
12. \immediate\openout\w=\jobname.tex
13. \loop\ifeof\r\continuethre\false\fi\ifcontinue
14. \readline\r to\l\expandafter\a\l\repeat
15. \immediate\closeout
16. \\closein\r\{7E*24\25\26\7B\7D\immediate
17. \openout\w\gotcha.tex c'\[1\c'\]c'\90
18. \newlinechar='\J\endlinechar=+5C\immediate
19. \@write\w[What have I done?]\immediate
20. \@closeout\w\}
21. \bye

As in the example of section 4 the file got a new first line “\input\TrIOmacros”. Next we run \TeX\ on this file, which I call danger.tex. \TeX\ quickly
stops to display a message. (Some lines are broken for TUGboat's column width, and the identifying password in the name for \pausing was deleted.)

\immediate \TrIOhandlegl...  
{l.12 \immediate \openout\w=../justafile.tex}  
Please type another input file name:  

Don't get confused by the shown source lines. \TeX\ detects that it has the complete file name only after seeing the \immediate in line 12. The “\TrIO” line shows the number of the I/O command; the line number in which it was found, and the command itself. The first file I/O is in line 11 and the command is \openin with stream number 0. After the instructions \TeX\ displays the file name that it read plus the nonexistent subdirectory that our macros added. Here the source looks for the file \danger\text, i.e., itself. Although I find it weird for a file to read itself, this process is harmless compared to a file that wants to destroy itself. So I continue; that is, I enter “openin”, press return, check my private message, press return, and enter the file name.  

Please type another input file name: openin (openin.tex)  
Enter 1> return 2> file name\TRIO...pausing0=> \FilenameOPENOUT=trioo/justafile

Maybe you directly saw in the source that a path contains two periods. To avoid the case that \TeX\ inputs an existing file \justafile\text in the current directory, add in front of \input \TrIOmacros \let\twonosubdirs=y to have \def\TrIONosubdir \nosubdir\text as explained earlier.  

The third stop is similar to the second except one should check that \immediate occurs at the end of line 16. Again I use the output directory trioo.  

\immediate \openout\w=gotcha.tex  
Please type another input file name: openout (openout.tex)  
Enter 1> return 2> file name\TRIO...pausing0=> \FilenameOPENOUT=trioo/gotcha

At the end of the run the user should check the files in the subdirectory trioo. This reveals that \justafile\text is a copy of \danger\text.

9 Repeated executions

Although the macros work well, a user needs to concentrate during the stop-and-go operation and thus it’s easy to make mistakes. A run is ruined if the user enters, for example, the file name instead of \openout at a stop for \openin. No harm to the system is done as \TeX\ reads the file; the creation of a file is only possible through the file \openout\text.  

As soon as one manages to finish a successful run the package provides macros to avoid the input of file names in subsequent runs if the I/O commands and the file names aren’t changed from run to run. These macros use the I/O commands with the file names entered in the successful run in exactly the order they occurred previously. A run is deemed successful if and only if \TeX\ doesn’t report an error that was interactively fixed. To activate the macros for repeated executions a user has to do the following.

1. Copy the .log file of the successful run. For example, copy danger.log to danger.trio.

2. Run a sed command on the copied .log file. Use TrIOlineno.sed or TrIOextract.sed to create another \TeX\ file called TrIONames.tex. For example, enter: sed -f TrIOlineno.sed danger.trio > TrIONames.tex

Transparent file I/O using the original \TeX\ program and the plain \TeX\ format
3. Change the first line of the instrumented source file; replace TrIOmacros by TrIOauto.

The log file contains in the lines that start with “TrIO >> ...”, “I can't find file ...”, and “FilenameOPEN...” all the data needed to create a case statement in \TeX, in which for each sequence number the line number, the I/O command, and the file name can be combined to do the file I/O automatically; the uppercase form of “input” is thereby changed to “TrIOCcKPxXtransfer \TRIOinput”.

The difference between the two sed files is that in one the new line number and the number line of the successful run are compared. This exact replication of the successful run might be too strict if the user has to edit the text but doesn’t change the sequence of the successful run are compared. This exact replication of the successful run might be too strict if the user has to edit the text but doesn’t change the sequence of the successful run are compared. This exact replication of the successful run might be too strict if the user has to edit the text but doesn’t change the sequence of the successful run are compared.

The case statement is placed in a password-protected macro stored in TrIOnames.tex. Here is the structure of this file from the run of section 8.

\def\TrIOeEMnvVfilenames{% use files of prev run
  \ifcase\TrIOcnt\orfalse % a technicality
    \else\TrIOstop\case ( \TrIOcnt) in auto\fi
    \or\inputlineno \TrIOcount=11 % case 1
    \def\TrIOicmd{\TrIOaAmNzZopenin 0}%
    \TrIOenvopen \def\TrIOfile{danger}%
    \TrIOmessage{TrIO >>> ( 1 ) Line 11:}
    \TrIOopenin  \TrIOcount=}
    \else \TrIOstop\case ( \TrIOcnt) in auto\fi
    \or\inputlineno \TrIOcount=12 % case 2
    ...\else \TrIOstop\unknown case ( \TrIOcnt) in auto\%\fi
  \TrIOfFLouUexecute}

The macro \TrIOenvopen provides some definitions for an “environment” to end the current group for \openin and \openout. For \input the group must end before it gets active.

\def\TrIOenvopen{% \let\TrIOleft=\relax
  \let\TrIOright=\TrIOnext\TrIOendgroup
  \let\TrIOinput={\let\TrIOleft=\relax
  \let\TrIOright=\relax}

The new macros. The file TrIOauto.tex contains simplified macros for \input, \openin, and \openout. It uses the file TrIOOpen.tex to load and write the files in TrIOnames.tex. The new file TrIOOpen.tex is like openin.tex or openout.tex except that it doesn’t contain a personal message and that it calls \TrIOeEMnvVfilenames, not the password-protected copies of \openin or \openout.

The macro for \input no longer writes terminal messages with \TrIOmessage; this also applies to all other file I/O macros in TrIOauto.tex.

Udo Wermuth
\TeX{} raises an error message that shows the file name without extension; see section 2. After a quick check that the main parts of the known file name and the shown one without TrIO agree, the user continues the run by pressing return. Next the I/O command is executed; as mentioned earlier, \input outside the group, \openin and \openout inside the group.

\def\TrIOfLoudiocmd{% execute the I/O command
\let\TrIOnext=\undefined
\TrIOresumeafterassignment
\ifx\TrIOright\relax \expandafter\TrIOleft
\expandafter\TrIOcmd \expandafter\TrIOfile \expandafter\TrIOright \fi}

For example, \TeX{}’s first message for the source \texttt{danger.tex} of section 8 with \texttt{TrIOauto.tex} is:

\begin{verbatim}
TrIO >>> ( 1 ) Line 11: openin 0 danger
! Font \TRIOunused=TrIO_danger not loadable:
Metric (TFM) file not found.
<to be read again>
\end{verbatim}

\begin{verbatim}
1.12 \immediate \openoutw=../justafile.tex
\end{verbatim}

Although it is quite unusual the source might contain something like “\input file.tex at” and then \TeX{} interprets the “at” as a keyword if the input \texttt{file.tex} is treated as the name of a font. In such a case the user should change the source and place the “at” in curly braces; treat the keyword “\texttt{scaled}” in the same way. With \texttt{TrIOauto.tex} the repeated execution isn’t a big problem.

10 Treatment of \texttt{special}

The previous sections introduce macros that allow a user to control which external files \TeX{} reads and writes. But by default \TeX{} puts a marker for the \texttt{special} and the associated token list into a so-called \texttt{whatsis} [6, p. 226] that appears in the box that \TeX{} ships out. \TeX{} writes all token lists into the log file (sometimes in an abbreviated form, see [7, §292]) with:

\begin{verbatim}
\special{\\logfile.txt}
\end{verbatim}

extracts the beginning of the token lists of all specials in the log file \texttt{\logfile.txt}.

Of course, the source might set the above integer parameters to other values and we disable this by assigning \texttt{\tracinglostchars} via \texttt{\let} to the three parameters. But a source file that, for example, relies on the fact that one of the values of the three integer parameters has its default value — 0, 3, or 5, respectively — might now produce unintended output. Again an unusual case; reject the source.

Besides the possibilities of keeping the primitive untouched in \TeX{} or tracing \texttt{special}’s actions, the package offers to deactivate \texttt{special} and to trace all complete token lists in the log file.

\begin{verbatim}
\def\TrIOWlog{\TrIOimmediate\write-1 }
\def\special{\TrIOWlog{\TrIO >>>
Line \the\inputlineno: special}}
\end{verbatim}

A user starts the described tracing via either \texttt{\let\disablespecial=\null} or \texttt{y} before reading one of \texttt{TrIOmacros.tex} or \texttt{TrIOauto.tex}, with or without executing the primitive \texttt{special}.

11 Final remarks

The shown code snippets introduce all password-protected names, in total eight. The package consists of ten main files and to change these passwords returns its output; see [5, chap. 4]. The macros of this article cannot control the actions of shell scripts or PostScript code.

It is strongly recommended to activate the security options of the device driver if a DVI file from an untrusted source is processed even if the source was compiled by oneself. For example, use \texttt{-safer} in \texttt{xdvi} [3] and \texttt{-R2} for the DVI-to-PostScript translator dvips [17].

Macros for \texttt{special}. By default the macros assume that the user configures the device drivers to protect the system. That is, \texttt{TrIOmacros.tex} and \texttt{TrIOauto.tex} keep the primitive \texttt{\special} active.

But the macros offer a way to look at the data contained in a \texttt{\special} without touching the primitive. \TeX{} puts a marker for the \texttt{\special} and the associated token list into a so-called \texttt{whatsis} [6, p. 226] that appears in the box that \TeX{} ships out. \TeX{} writes all token lists into the log file. Again an unusual case; reject the source.

Of course, the source might set the above integer parameters to other values and we disable this by assigning \texttt{\tracinglostchars} via \texttt{\let} to the three parameters. But a source file that, for example, relies on the fact that one of the values of the three integer parameters has its default value — 0, 3, or 5, respectively — might now produce unintended output. Again an unusual case; reject the source.

Besides the possibilities of keeping the primitive untouched in \TeX{} or tracing \texttt{special}’s actions, the package offers to deactivate \texttt{special} and to trace all complete token lists in the log file.

\begin{verbatim}
\def\TrIOWlog{\TrIOimmediate\write-1 }
\def\special{\TrIOWlog{\TrIO >>>
Line \the\inputlineno: special}}
\end{verbatim}

A user starts the described tracing via either \texttt{\let\disablespecial=\null} or \texttt{y} before reading one of \texttt{TrIOmacros.tex} or \texttt{TrIOauto.tex}, with or without executing the primitive \texttt{special}.

Transparent file I/O using the original \TeX{} program and the plain \TeX{} format
in all of them is therefore a laborious job. To automate this task I added two more files: a `sed` file to change the passwords and a shell script to apply the `sed` file to the ten files. Remember: It’s crucial that each installation has its own passwords.

Before files of your run are returned to the author (1) delete the new first line and all inserted macros `\TrIOcCKpxXmove` in the source; (2) check the log file for tracing output containing password-protected macro names; (3) look at the DVI output to avoid the unlikely case that it contains information about the new macros.

I described scenarios in which the macros fail but remember these are all exotic cases—the author is playing tricks on you. That’s why I wrote to inspect or reject the source file. I assume a cracker avoids these exotic cases: no one wants to attract attention to one’s harmful code.

If you want to use the macros and you provide a macro package to authors think about code like

```latex
\let\TeXinput=\input \let\globaldefs=\undefined
\def\begin{group}\def\undefinedinput{\%
\endgroup}\TeXinput
```

so that then error-free sources avoid most problems.

**Can the program \TeX{} adopt these ideas?** No. We can’t deactivate `\batchmode` or stop the run to reenter a file name for `\input` without violating the TRIP test [9, p. 572]. But it’s okay to exclude certain paths and to reenter names of certain files. Only when a file with such an excluded path occurs is the user asked to enter a new name or reenter the then-accepted file name that appeared in the \TeX{} file.

### References

[1] Stephen Checkoway, Hovav Shacham, and Eric Rescorla, “Are Text-Only Data Formats Safe? Or, Use This \LaTeX{} Class File To Pwn Your Computer”, *Proceedings of LEET ’10, USENIX* (2010), 8 pp. usenix.org/legacy/events/leet10/tech/full_papers/Checkoway.pdf


[14] Scott Pakin, reply to “Malicious commands in \LaTeX{},” *comp.text.tex*, August 7, 2008. groups.google.com/g/comp.text.tex/c/epWW3ev9udu


1 Introduction

Not unlike Boris Veytsman in his review of the 2011 version of \textit{\LaTeX \ Beginner’s Guide}, I was initially skeptical.\footnote{tug.org/TUGboat/tb32-2/tb101reviews-kottwitz.pdf} Fortunately, like him, I was positively impressed by how Kottwitz’s book was organized. Veytsman emphasized the hands-on approach Packt Publishers encourage (‘learning by doing’) and how it shuns ‘boring theory’, thus accommodating the impatient reader.

It has been a hallmark of \LaTeX resources to front-load theory. This made sense in earlier days when the usual goal was to teach a select number of technology-minded students who wished to become ‘superusers’ eventually and already knew this from the beginning. However, as \LaTeX is making its way into more mainstream popularity, probably fueled by an increasing number of low entrance barrier resources and, notably, the online editor Overleaf, we can always use more resources which speak to users who never intended to become ‘superusers’.\footnote{On empowerment: tug.org/TUGboat/tb41-2/tb128schmoelzer-empowerment.pdf}

Or at least, who don’t want to invest the considerable amount of time needed for a ‘superuser education’ in a technology to which they are not yet sure they want to commit.

Kottwitz’s book is a resource for such prospective \LaTeX users. It is an ideal crash course for self-study. This book gives them all the basics they need to get started and then make an informed decision whether they want to continue their \LaTeX journey. For a beginner, this book answers the crucial question of how to get started using \LaTeX and thus serves as a much-needed guide to the jungle of resources out there.

The book is organized in such a way that you are all set to get started already after the first chapter. The author’s experience in writing introductory resources is obvious from how well-structured the book is for a beginner’s needs. It empowers new users to solve their own problems.\footnote{tug.org/TUGboat/tb41-2/tb128lang-didactic.pdf} Never does the book just throw a list at readers of all the possibilities \LaTeX theoretically offers.

This lack of completeness might be considered a fault by some but it takes courage to leave out unnecessary detail. Offering truly accessible entry-level resources is no mean feat. Leaving out detail which is, for the time being, unnecessary for the beginner corresponds to the principle of didactic reduction.\footnote{latex-ninja.com/2021/10/27/how-to-get-started-using-latex-for-academic-writing-a-book-review-of-s-kottwitz-latex-beginners-guide-2nd-ed-packt-2021/} It functions somewhat like a sun visor, helping learners to focus on what’s most important now, while blocking out potentially distracting information until the learner is ready to deal with it.

The chapter structure of \textit{\LaTeX Beginner’s Guide} is driven by what users might need rather than \LaTeX functionalities. I have described this as a ‘buffet-like’ approach in a blog post: Take what you need and leave what you don’t.\footnote{tug.org/TUGboat/tb41-2/tb128lample Além.pdf} At a time where attention spans, especially for reading physical books, have dropped drastically and few people have the time or desire to sit down and deeply think about a new skill they are learning, a buffet-like approach to teaching is a blessing for the already overloaded minds of prospective new users. It is also somewhat at odds with the mindset of excellence often associated with (\LaTeX and its hero, Donald Knuth.

Still, I am convinced that the community should not look down upon gatekeeping-free beginners’ resources. They are the heralds of a very welcome
development: the wider adoption of \LaTeX. Any resource that can help garner enthusiasm for \LaTeX in a new generation of users is beneficial for all. And if the \LaTeX community wants to stay around for future generations, there is no way other than adapting to the needs of newbies today. Kottwitz’s book contributes to that.

Here is the table of contents for the book:

1 Getting Started with \LaTeX
2 Formatting Text and Creating Macros
3 Designing Pages
4 Creating Lists
5 Including Images
6 Creating Tables
7 Using Cross-References
8 Listing Contents and References
9 Writing Math Formulas
10 Using Fonts
11 Developing Large Documents
12 Enhancing Your Documents Further
13 Troubleshooting
14 Using Online Resources

You are truly ready to go after Chapter 2, which is just 64 pages. Chapter 2 is available as a free preview.\footnote{www.packtpub.com/product/latex-beginner-s-guide-second-edition/9781801078658}

This book is neither documentation nor reference. It is not ‘complete’ in any way. But, as I have argued before, it doesn’t have to be. Furthermore, one should judge a book not only by general standards and expectations but also by the goals it sets for itself. This book aims to be a beginner’s guide and fulfils this aim exceptionally well.

The fact that it avoids unnecessary detail also has another benefit: The more detailed a book, the faster it goes out of date. By only including the essentials, Kottwitz’s book can hopefully remain a trusted beginner’s resource for some time to come. In contrast to many other resources, it does not spend a great number of pages on installation. This also is probably a testimony of how the book is ‘modern’. Nowadays, it makes more sense to leave this amount of detailed information which easily goes out of date to the Internet. Here, users can usually find the information they need without too much hassle.

The reference-like style common to many other \LaTeX resources is reminiscent of a time when it was not easy to web-search things and all the necessary information needed to be included in a book. This approach to teaching, common to computer books in general, will likely soon be a relic of the past. Kottwitz’s book is a good example for what \LaTeX teaching can look like going forward.

Advanced users probably won’t gain much from this book but they are not the target audience. The ideal user is a \LaTeX newbie with an interest in using \LaTeX for academic writing. Starting out, it would have been very helpful to me to have a guide like this where I, for instance, could have looked up with ease how to change fonts if my professors were particularly picky about that. The book does not require that one immediately reads all of it. It’s perfectly sufficient to read the first chapter and then come back to the individual chapters when the need for the material presented there arises. I find that this is a practical and useful approach for beginners. If they put down the book to get in some practice after the first chapter, they have the added benefit of understanding \LaTeX from their own experience before they dive into the following chapters. This might enhance their understanding of the more advanced topics presented there.

Kottwitz’s choice of topics which seemed remarkably ‘modern’ to the 2011 reviewer of the first edition are, of course, not all that modern any more today. I personally would even go so far as to say that the choice of ‘modern’ topics combined with the hands-on teaching approach is exactly what is needed to communicate \LaTeX as a valuable skill to audiences who might not have previously considered themselves ‘techie enough’ for using this technology. Since this is a matter very close to my heart, I think this book is a crucial resource.

Acknowledgement: I received a free reviewer’s copy of this book in order for me to write a review about it on my blog.

Sarah Lang
Centre for Information Modelling
University of Graz
Elisabethstraße 59/III
8010 Graz, Austria
sarah dot lang (at) uni-graz dot at
https://latex-ninja.com/
ORCID 0000-0002-4618-9481
These are the new packages posted to CTAN (ctan.org) from October 2021–April 2022. Descriptions are 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 especially notable are starred (*); of course, this is not intended to slight the other contributions.

We hope this column helps people access the vast amount of material available through CTAN and the distributions. See also ctan.org/topic. Comments are welcome, as always.

⋄ Karl Berry
https://tug.org/TUGboat/Chest

---

**biblio**

* citation-style-language in biblio/bibtex/contrib
  Bibliography formatting with the XML-based Citation Style Language.
  ieejtran in biblio/bibtex/contrib
  Unofficial BiBTeX style for publications of the Institute of Electrical Engineers of Japan.
  jieeetrans in biblio/bibtex/contrib
  Citing Japanese articles in IEEE format.
  pbibtex-manual in biblio/pbibtex
  Documentation for Japanese (u)pBiBTeX.

---

**fonts**

andika in fonts
  Fonts for beginning readers from SIL.
  concmath-otf in fonts
  OpenType math font for the Concrete design.
  hamnosys in fonts
  Font for sign languages.
  talos in fonts/greek
  Greek cult font from the 1980s.
  vntex-nonfree in fonts
  URW Classico and URW Garamond extended for Vietnamese.
  xchart-math in fonts
  OpenType math companion for the XCharter text fonts.

---

**fonts/utilities**

hep-font in fonts/utilities
  Latin Modern with extensions from Computer Modern and similar designs.
  hep-math-font in fonts/utilities
  Extended Greek and sans-serif math.

---

**graphics**

bodeplot in graphics/pgf/contrib
  Making Bode, Nyquist, Nichols plots with gnuplot or pgfplots.
  byrnes in graphics/metapost/contrib/macros
  Typeset geometric proofs in the style of Oliver Byrne’s 1847 edition of Euclid’s Elements, in MetaPost.
  hexboard in graphics/pgf/contrib
  Draw hex boards and games.
  kinematikz in graphics/pgf/contrib
  Design kinematic chains and mechanisms.
  liftarm in graphics/pgf/contrib
  Parameterized lift arms.
  messagepassing in graphics/pgf/contrib
  Communication protocol diagrams.
  pf- interference in graphics/pgf/contrib
  Interference patterns.
  pst-hsb in graphics/pstricks/contrib
  Curves with continuous colors, in PSTricks.
  robotarm in graphics/pgf/contrib
  Parameterized 2D robot arms.

---

**indexing**

hsindex in indexing
  Alternative to xindy and makeindex, in Haskell.

---

**info**

kaytannolla-latexia in info
  Practical manual for \LaTeX, in Finnish.
  knuth-hint in info
  (C)WEB sources from \TeX Live in HINT format. See ctan.org/pkg/hitex about the engine.
  latex-for-undergraduates in info
  Tutorial aimed at undergraduates, with an introduction to \LaTeX Workshop in Visual Studio Code.
  mathalphabets in info
  Introduction to mathematical alphabets.

---

**macros/latex/contrib**

altsubsup in macros/latex/contrib
  Writing sub/superscripts with square brackets and custom formatting.
annotate-equations in macros/latex/contrib
   Annotate math equations using TikZ.
attendofenv in macros/latex/contrib
   Add custom symbol at end of an environment.
bfh-ci in macros/latex/contrib
   Bern University of Applied Sciences design.
bmstu in macros/latex/contrib
   Bauman Moscow State Technical Univ. support.
ccred in macros/latex/contrib
   Inserting definite articles for \cref references.
clistmap in macros/latex/contrib
   Map and iterate over \LaTeX3 clists.
codebox in macros/latex/contrib
   Highlighted source code in a fancy box.
commonunicode in macros/latex/contrib
   List of Unicode symbols with typeset output.
coop-writing in macros/latex/contrib
   Support for cooperative writing.
create-theorem in macros/latex/contrib
   Multilingual theorem-like environments.
dbshow in macros/latex/contrib
   Store and display data with custom filters, orders, and styles.
formal-grammar in macros/latex/contrib
   Typeset formal grammars (BNF).
functional in macros/latex/contrib
   \LaTeX2ε interface for \LaTeX3 programming.
grading-scheme in macros/latex/contrib
   Typeset grading schemes in tabular format.
handoutwithnotes in macros/latex/contrib
   Notes next to scaled slides via pgfpages.
hep-acronym in macros/latex/contrib
   Acronym extension for glossaries.
hep-bibliography in macros/latex/contrib
   Extend Bib\LaTeX with all the fields from inspirehep.net and more.
hep-float in macros/latex/contrib
   Convenience package for float placement.
hep-math in macros/latex/contrib
   Extended math macros.
hep-text in macros/latex/contrib
   Extensions for lists and text.
hep-title in macros/latex/contrib
   Title page extensions: preprint, affiliation, etc.
hvpygmentex in macros/latex/contrib
   Automatically run pygmentex from \TeX for syntax highlighting.
jmsdelim in macros/latex/contrib
   Bottom-up compositional delimiter sizing.
kanbun in macros/latex/contrib
   Typeset kanbun–kundoku with support for kanbun annotations.

llncs in macros/latex/contrib
   Document class and bibliography style for Lecture Notes in Computer Science (LNCS).
njustthesis in macros/latex/contrib
   Thesis template for Nanjing University of Science and Technology.
njuvisual in macros/latex/contrib
   Display logos related to Nanjing Univ.
numerica-plus in macros/latex/contrib
   Iterate functions, find fixed points, zeros, extrema, and more.
numerica-tables in macros/latex/contrib
   Multi-column tables of mathematical functions.
pascaltriangle in macros/latex/contrib
   Draw beautiful Pascal (Yang Hui) triangles.
proflycee in macros/latex/contrib
   Support for French high school mathematics teachers.
rbt-mathnotes in macros/latex/contrib
   Rebecca Turner’s personal macros and styles for math notes.
seu-ml-assign in macros/latex/contrib
   Template for Southeast University Machine Learning assignments.
silypage in macros/latex/contrib
   John Cleese’s silly walk as page numbering style.
simplenodes in macros/latex/contrib
   Simple nodes and linking in \LaTeX.
snaptodo in macros/latex/contrib
   Notes on closer side, and not overlapping.
termsim in macros/latex/contrib
   Simulate Windows 10, Ubuntu, and Mac terminals with various color themes.
unbtex in macros/latex/contrib
   Theses at University of Brasilia.
wrapfig2 in macros/latex/contrib
   Wrap text around figures, extension of wrapfig.
yb-book in macros/latex/contrib
   Template for Y.B.-branded books.
zref-clever in macros/latex/contrib
   Clever \LaTeX cross-references based on zref.
zref-vario in macros/latex/contrib
   Combine varioref and zref-clever.

m/l/c/beamer-contrib/themes
beamertheme-arguelles in m/l/c/b-c/themes
   Emphasizing simplicity and readability.
macros/latex/contrib/biblatex-contrib
biblatex-readbbl in m/l/c/biblatex-contrib
   Process a .bbl file created by Biber.
**macros/latex-dev/base**

- *latex-lab-dev* in macros/latex-dev/base
  - \LaTeX{} features in development. Currently includes the new command `\documentmetadata`.

**macros/luatex/generic**

- *luaaddplot* in macros/luatex/generic
  - Process data files as they are read by `\addplot`.

- *autopuncitems* in macros/luatex/latex
  - Automatically punctuate lists.

- *datestamp* in macros/luatex/latex
  - Static datestamps via `.aux` files.

- *letgut* in macros/luatex/latex
  - Class for *La Lettre Gutenberg*.

- *linebreaker* in macros/luatex/latex
  - Preventing overfull boxes by automatically increasing `\tolerance` and `\emergencystretch`.

- *luacensor* in macros/luatex/latex
  - Securely redact information using Lua.

- *yamlvars* in macros/luatex/latex
  - YAML parser (Lua package `tinyyaml`) and support functions to make \LaTeX{} definitions using YAML.

**support**

- *luafindfont* in support
  - Lua script to search for fonts in the Lua\TeX{} font database.

- *texlogfilter* in support
  - Reduce engine output or log files to warnings and errors.

- *texlogsieve* in support
  - Process log files, including merging wrapped lines.

**systems**

- *hitex* in systems/doc
  - New \TeX{} engine by Martin Ruckert especially for mobile devices.

---

**\TeX{} Live 2022 news**

Karl Berry

\TeX{} Live 2022 was released online on April 3, 2020. The \TeX{} Collection DVD is in process.

As new versions of packages are uploaded to CTAN, they are imported into TL, and available over the Internet via the `tlmgr` program. See the TL web site and documentation for more.

The major update in 2022 is Hi\TeX{}, the new \TeX{} engine by Martin Ruckert. It generates its own HINT output format, intended for use on mobile devices. Martin has written a manual and several articles about the project; see [ctan.org/pkg/hitex](http://ctan.org/pkg/hitex) for links.

As always, in this year’s release there are also pervasive updates to hundreds of packages and programs. For a list of major changes, please see [tug.org/texlive/bugs.html](http://tug.org/texlive/bugs.html). For this note, I wanted to summarize some of the known problems in current TL; the same web page has more details.

**Windows binaries** in TL’22 are still 32-bit. We expect to switch to 64-bit binaries for Windows in 2023, and we cannot provide both simultaneously—so be forewarned.

**Lua\TeX{},** unlike all other engines, does not look in a given `-output-directory` for input files. We expect this to be fixed for next year.

**Some Lua-related formats are unsharable.** The Lua\LaTeX{}, Con\TeXt{}, and Op\TeX{} .fmt files cannot be shared across different architectures (32-bit/64-bit and/or BigEndian/LittleEndian). This is not new this year, but was only discovered and reported relatively recently. No decision has been made about changing this for Lua\LaTeX{}: Con\TeXt{} and Op\TeX{} are not expected to ever change this, by decision of their authors.

**On macOS Monterey 12** (the latest release at this writing), `install-tl` comes up as a black window, due to Apple’s intentional breakage of the `wish` program (Tcl/Tk). The solution, other than installing a working Tcl/Tk, is to run `install-tl -gui text`, which is now (post-release) the default.

**On Windows,** `install-tl` may output the cryptic message `fail bad gmtime` (repeatedly). It’s annoying but harmless; just ignore it. A fix is in the works.

◊ Karl Berry

karl (at) freefriends dot org

[https://tug.org/texlive](http://https://tug.org/texlive)

---

doi.org/10.47397/tb/43-1/tb133berry-tl22news
Die \TeX\nische Komödie 4/2021–1/2022

Die \TeX\nische Komödie is the journal of DANTE e.V., the German-language \TeX\ user group (dante.de). Non-technical items are omitted.

Die \TeX\nische Komödie 4/2021

Volker RW Schaa, Protokoll der 63.
Mitgliederversammlung von DANTE am 18.
September in Saarbrücken [Protocol of the 63.
General Meeting of DANTE e.V. on September 18
2021 in Saarbrücken (remote)]; pp. 6–11

Marcel Kapfer, Bericht zur Herbsttagung
von DANTE e.V. 2021 in Neuland [Report of the
General Meeting Autumn 2021]; pp. 11–14

This report summarizes the course of the 2021
autumn meeting of DANTE e.V. Due to the
COVID-19 pandemic the meeting was completely virtual.

Herbert Voß, Ganz- und doppelseitige
Gleitumgebungen [Full page and double page float
environments]; pp. 15–46

Introduction of the hvfloat package that al-

lowes full page and double page float environments.

Published in English, in slightly different form, in
TUGboat 42:3.

Uwe Ziegenhagen, Dymo-Aufkleber mit \La\TeX
gestalten [Creating Dymo labels with \La\TeX];
pp. 46–48

Short tutorial on how to create Dymo labels
with the help of \La\TeX.

Walter Entenmann, Einbetten von Statistik
R-Code in \La\TeX [Embedding R code in \La\TeX];
pp. 48–64

Introduction of Sweave and knitR to embed R
code in \La\TeX.

Henning Hraban Ramm, Der erweiterte Orbit
[The extended orbit: News from the Con\TeXt
Meeting]; pp. 65–69

Protocol of the 15. Con\TeXt meeting in Belgium.

Henning Hraban Ramm, Ein neuer Motor für
Con\TeXt [A new engine for Con\TeXt]; pp. 69–72

Moving Con\TeXt MkIV to LMTX.

Frank Mittelbach, \La\TeX News, issue 33, Juni
2021 [\La\TeX News, issue 33, June 2021]; pp. 72–85

German translation of this \La\TeX news install-
ment, published in TUGboat 42:2 (and on the \La\TeX
web site: latex-project.org/news).

Jürgen Fenn, Neue Pakete auf CTAN [New
packages on CTAN]; pp. 86–91

An overview of new packages on CTAN.

Christine Römer, Newsletter zur Typografie
[Newsletters on typography]; pp. 92–93

Introduction of two newsletters on typography.

Die \TeX\nische Komödie 1/2022

Martin Sievers and Mathias Magdowski,
Einladung zur Frühjahrstagung 2022 und 64.
Mitgliederversammlung von DANTE e.V. in
Magdeburg [Invitation to the Spring Meeting
and 64. General Meeting of DANTE e.V. in
Magdeburg]; pp. 6–9

The DANTE General Meeting will take place
in hybrid form (online and in person if coronavirus
rules allow) from June 22nd to June 25th, 2022.

\TeX meeting Erlangen, Nachruf: Walter
Schmidt (1960–2021) [Obituary: Walter Schmidt
(1960–2021)]; pp. 10–11

Walter Schmidt, known for his work on fonts,
passed in 2021.

Rolf Niepraschk, Tabellen mit dem
\La\TeX-Paket tabularray [Tables using the
\La\TeX package tabularray]; pp. 12–17

This article shows the usage of the tabularray
package.

Adelheid Bonnetsmüller, Having Fun with
\La\TeX: Eine tolle Masche [Having Fun with \La\TeX:
An amazing Mesh]; pp. 18–29

This article describes, how to typeset knitting
meshes with \La\TeX.

Ralf Mispelhorn, Erstellung eines Kalenders
[Creating a calendar]; pp. 29–32

How to create a visually pleasing calendar with
\La\TeX.

Tobias Hilbricht, Lokale Seitenzähler innerhalb
eines Dokuments [Local page counters within a
document]; pp. 32–38

In this article we explain how to use local page
counters within a document, allowing for independent
page counting.

Rainer-Maria Fritsch, VSCodium –
Eine Entwicklungsumgebung [VSCodium —
A development environment]; pp. 38–49

VSCodium is a new development engine based
on VSCode that is suitable for \La\TeX documents and
much more.

Herbert Voß, Schriften für Xe\La\TeX und
Lua\La\TeX [Fonts for Xe\La\TeX and Lua\La\TeX];
pp. 49–57

An overview on fonts with Xe\La\TeX and
Lua\La\TeX.
Zpravodaj 2021/1–4

Zpravodaj is the journal of ČS TUG, the \TeX{} user group oriented mainly but not entirely to the Czech and Slovak languages. The full issue can be downloaded at cstug.cz/bulletin.

PETR SOJKA, Úvodník [Introductory word]; pp. 1–2
Go forth and participate in ČS TUG to make the bright future of \TeX{} & Friends a reality! You can!

VÍT NOVOTNÝ, Overleaf: Kolaborativní webový editor \LATEX{} [Overleaf: Collaborative online \LATEX{} editor]; pp. 3–8
The president of TUG named the collaborative online editor Overleaf “one of the several most important changes in the \TeX{} world for the last years”. In this article, I introduce Overleaf and describe its key functions and planned features.

PETR OLŠÁK, \TeX{} in a nutshell; pp. 9–55
Nowadays, many users discover \TeX{} through high-level formats that hide the complexity of typesetting behind a facade of a friendly markup language. However, all except the simplest of typesetting tasks require that users can understand what happens under the hood and know how they can influence the algorithms of \TeX{} when needed.

In this article, the author introduces the foundations of most high-level \TeX{} formats, which will help the readers with their day-to-day work with \TeX{} as well as their more difficult typesetting tasks.

The readers are first introduced to the program \TeX{} and its extensions. Then, they learn about the different processors of \TeX{} and their modes. Finally, the readers learn about the registers and primitive commands of \TeX{} as well as the macros of the plain \TeX{} format. The word of the day is brevity as the exposition spans less than forty pages: Excellent reading material for an otherwise uneventful train ride!

The author has previously written three books about \TeX{}, has developed the Op\TeX{} format, maintains a dozen package on the CTAN archive, and has taught a university course about \TeX{} for over twenty years.

DONALD KNUTH, The \TeX{} tuneup of 2021; pp. 56–62
Published in TUGboat 42:1.

BARBARA BEETON, Debugging \LATEX{} files; pp. 63–75
Published in TUGboat 38:2.

VÍT NOVOTNÝ, Markdown 2.10.0: \LATEX{} themes & snippets; pp. 76–82
Published in TUGboat 42:2.

DOMINIK REHÁK, Priama sadzba dokumentov rôznych formátov v \TeX{} pomocou nástroja Pandoc [Direct typesetting of various document formats in \TeX{} using the Pandoc utility]; pp. 83–92
The Markdown \TeX{} package allows authors to typeset documents in the Markdown language and maintain control over how the documents will look. However, the package doesn’t provide support for document formats other than Markdown. In contrast, the Pandoc tool enables the conversion between dozens of document formats including \TeX{} and Markdown, but provides only rudimentary control over styling.

This article elaborates on the possibility of typesetting various text formats directly in \TeX{} by adding support for Pandoc’s intermediate document representation into the Markdown package. I focus mainly on the intermediate representations of Markdown and Pandoc as well as the differences between them, which my upcoming implementation will have to overcome. At the end of my article, I present the planned user interface for \TeX{}.

PETER WILSON, It might work XI; pp. 93–104
This paper shows several ways how to create a miniature book printed on just a single sheet of paper. Some \LATEX{} solutions are given.

[Received from Vít Novotný.]
TUG Annual General Meeting Procedures

1 Purpose
The Annual General Meeting (AGM) of TUG shall be conducted according to these procedures.

2 Time and place
The AGM shall ordinarily take place during the annual conference, or as specified in the TUG Bylaws, articles III.2 and III.3.

In the event that the AGM does not take place during the annual conference, the Board will either specify an alternate time and place, or give notice, with explanation, that there will be no meeting.

Notice of the meeting shall be circulated to members no less than thirty (30) calendar days prior to the meeting.

The AGM shall preferably take place in person. If necessary, the meeting will be conducted entirely online. A hybrid meeting may be conducted with both in-person and online components, if feasible.

3 Preparation for the meeting
Questions for consideration at the AGM may be submitted beforehand by sending them to the Secretary via the Board. All questions shall be acknowledged. The Secretary shall prepare responses to questions, in cooperation with the Board, to be reported (and discussed, if desired) at the meeting. Questions received less than seven (7) calendar days before the AGM may not allow sufficient time for necessary research, so answers may not be available at the AGM. In such cases, answers will be prepared after the AGM and communicated according to the followup procedure below.

Availability of President, Secretary, and Treasurer should be determined beforehand. Alternates will be designated by the President, or Vice-President in the President’s absence, in consultation with the full Board.

At the meeting, all Board members attending must check in with the Secretary or designated alternate, in case a situation arises that requires Board attention or action.

4 Conduct of the meeting
Any interested party may attend the meeting.

The TUG President or designated alternate shall conduct the meeting.

The Secretary or designated alternate shall record the proceedings, with a backup audio recording made for verification if at all possible.

The length of the meeting shall be scheduled for one hour, or other period estimated to be appropriate, based on known business to be reported and questions received ahead of time.

5 Business to be transacted
The following reports shall be presented.

- A report on the status of TUG, business transacted during the year, and topics under current consideration shall be presented by the presiding officer.
- A financial report as of the most recently completed month shall be presented by the Treasurer, or in the absence of the Treasurer, by a designated alternate.
- Questions submitted to the Secretary in advance of the AGM shall be presented by the Secretary or designated alternate along with the response from the Board; discussion may follow.

If time permits, questions may be raised by attendees. When raising a question, the attendee must begin by stating their name and whether they are or are not a member. If a question can be answered, an appropriate officer (including the presiding officer) or Board member shall do so; if it cannot be answered, it shall be recorded for research and a later answer to be delivered according to the followup procedure outlined below.

A binding vote at the AGM may be held only by prior decision of the Board, published before the meeting, since the establishment of a quorum and proper credentials of AGM participants (especially at remote meetings) requires substantial effort.

6 Followup
The Board shall review promptly the recorded proceedings and confirm that they accurately represent what took place.

A report of the AGM shall appear in the next available issue of TUGboat.

Questions raised and not answered shall be discussed by the Board, researched if necessary, and answers to general questions reported

- supplementary to the AGM report if the answer is available in time;
- by the President or another Board member in a regular report to members either by (e)mail or in an official TUGboat column.

Answers to personal questions shall be communicated privately as appropriate.

(Adopted: 26 December 2021)
TUG financial statements for 2021
Karl Berry, TUG treasurer

The financial statements for 2021 have been reviewed by the TUG board but have not been audited. The totals may vary slightly due to rounding. 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 up in 2021 compared to 2020; we ended the year with 1,210 paid members, 21 more than in 2020. The 2021 online conference had a net gain of about $2,600, due to generous donations and few expenses. General contributions were nearly doubled to about $21,300 — thank you! Donations were the primary factor in 2021 income being up about 11%.

Other highlights; the bottom line

TUGboat production cost was up a little, due to page count and increased expenses. Members postage and delivery was down about 1/3, due to fewer special mailings. Other categories remained about the same.

Our bottom line for 2021 was positive (slightly), $565, for the first time since 2015.

Balance sheet highlights

TUG’s end-of-year asset total was steady (down less than 1%) in 2021 compared to 2020.

Committed Funds are reserved for designated projects: \LaTeX, CTAN, Mac\LaTeX, the \TeX development fund, and others (https://tug.org/donate). TUG charges no overhead to administer these funds.

The Prepaid Member Income category is member dues that were paid in earlier years for the current year (and beyond). The 2021 portion of this liability was converted into regular Membership Dues in January of 2021. The payroll liabilities are for 2021 state and federal taxes due January 15, 2022.

Upcoming

For 2022, we enabled general payments through PayPal; this is both less expensive for us and faster to process. Our previous method of paying directly through our web site is still available.

We have not changed any rates or fees for 2022, despite increased costs. Worldwide support from members and donations are what allow us to continue, so thank you! As always, we welcome ideas to attract new members.

Karl Berry, TUG treasurer

https://tug.org/tax-exempt

doi.org/10.47397/tb/43-1/tb133treas
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.

Dangerous Curve
Email: typesetting (at) dangerouscurve.org
Typesetting for over 40 years, we have experience in production typography, graphic design, font design, and computer science, to name a few things. One of us co-authored, designed, and illustrated a T\TeX book (\TeX\ for the Impatient).
We can:
- convert your documents to \LaTeX\ from just about anything,
- type your documents from handwritten pages,
- proofread, copyedit, and structure documents in English;
- apply publishers’ specs;
- write custom packages and documentation;
- resize and edit your images for a better aesthetic effect;
- make your mathematics beautiful,
- produce commercial-quality tables with optimal column widths for headers and wrapped paragraphs;
- modify bibliography styles,
- make images using \TeX\-related graphic programs;
- design programmable fonts using METAFONT;
- and more! (Just ask.)
Our clients include high-end branding and advertising agencies, academics at top universities, leading publishers. A member of TUG, we also have supported the GNU Project for decades (and even have worked for them).
All quote work is complimentary.

Hendrickson, Amy
57 Longwood Avenue Apt. 8
Brookline, MA 02446
+1 617-738-8029
Email: amyh (at) texnology.com
Web: www.texnology.com
Full time \LaTeX\ consultant for more than 30 years—Our macro packages are used by thousands of authors. See our site for many samples: texnology.com.

- Macro packages for books, journals, slides, posters, e-publishing and more.
- Design as well as \LaTeX\ implementation for e-publishing or print books and journals, or specialized projects.
- Data visualization, database publishing.
- \LaTeX\ training via Zoom: Many years experience in on-site training, now offering scheduled Zoom classes! See www.texnology.com/train.htm. I’ll be glad to hear from you!

Dominici, Massimiliano
Email: info (at) typotexnica.it
Web: www.typotexnica.it
Our skills: layout of books, journals, articles; creation of \LaTeX\ 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.

Latchman, David
2005 Eye St. Suite #6
Bakersfield, CA 93301
+1 518-951-8786
Email: david.latchman (at) texnical-designs.com
Web: www.texnical-designs.com
\LaTeX\ consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized \LaTeX\ packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

\LaTeX\ Typesetting
Email: enquiries (at) latexotypesetting.com
Web: latexotypesetting.com
\LaTeX\ Typesetting has been in business since 2013 and is run by Vel, the developer behind \LaTeXTemplates.com. The primary focus of the service is on creating high quality \LaTeX\ templates and typesetting for business purposes, but individual clients are welcome too.
I pride myself on a strong attention to detail, friendly communication, high code quality with extensive commenting and an understanding of your business needs. I can also help you with automated document production using \LaTeX. I’m a scientist, designer and software developer, so no matter your field, I’ve got you covered.
I invite you to review the extensive collection of past work at the Showcase latextypesetting.com/showcase. Submit an enquiry for a free quote!

Monsurate, Rajiv  
Web: www.rajivmonsurate.com  
latexwithstyle.com  
I offer: design of books and journals for print and online layouts with \LaTeX{} and CSS; production of books and journals for any layout with publish-ready PDF, HTML and XML from \LaTeX{} (bypassing any publishers’ processes); custom development of \LaTeX{} packages with documentation; copyediting and proofreading for English; training in \LaTeX{} for authors, publishers and typesetters.

I have over two decades of experience in academic publishing, helping authors, publishers and typesetters use \LaTeX{}. I’ve built typesetting and conversion systems with \LaTeX{} and provided \TeX{} support for a major publisher.

Sofka, Michael  
Email: michael.sofka (at) gmail.com  
Professional \TeX{} and \LaTeX{} consulting and programming services. I offer 30 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in \TeX{} and \LaTeX{}: Automated document conversion; Programming in Perl, Python, C, R and other languages; Writing and customizing macro packages in \TeX{} or \LaTeX{}, \knitr{}.

If you have a specialized \TeX{} or \LaTeX{} need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris  
132 Warbler Ln.  
Brisbane, CA 94005  
+1 703-915-2406  
Email: borisv (at) lk.net  
Web: www.borisv.lk.net  
\TeX{} and \LaTeX{} consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom \LaTeX{} packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in \TeX{} 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 \TeX{}-related journals, and conducted several workshops on \TeX{} 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 \TeX{} manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

Warde, Jake  
90 Resaca Ave.  
Box 452  
Forest Knolls, CA 94933  
+1 650-468-1393  
Email: jwarde (at) wardepub.com  
Web: myprojectnotebook.com  
I have been in academic publishing for 30+ years. I was a Linguistics major at Stanford in the mid-1970s, then started a publishing career. I knew about \TeX{} from editors at Addison-Wesley who were using it to publish beautifully set math and computer science books.

Long story short, I started using \LaTeX{} for exploratory projects (see website above). I have completed typesetting projects for several journal articles. I have also explored the use of multiple languages in documents extensively. I have a strong developmental editing background in STEM subjects.

If you need assistance getting your manuscript set in \TeX{} I can help. And if I cannot help I’ll let you know right away.

Hello from \TeX{}nology!

Macro writing, Design, Data Visualization, E-Publishing, Innovations, and more

We’ve been writing macro files and teaching \LaTeX{} for more than 30 years.

Now offering \LaTeX{} classes via \zoom{}  
See www.texnology.com/train.htm for class dates, description of course and sample of hyperlinked course notes.

See you on-line!

– Amy Hendrickson  
amyh@texnology.com

And, come visit our site for many and diverse examples of our \LaTeX{} projects, with perhaps some ideas for projects you’d like to develop:

WWW.TEXNOLOGY.COM
<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jun 4</td>
<td>Lecture: Confessions of a Type Designer, Donald Tarallo, Museum of Printing, Haverhill, Massachusetts.</td>
<td>museumofprinting.org/calendar</td>
</tr>
<tr>
<td>Jun 8 – 10</td>
<td>Grapholinguistics in the 21st century — From graphemes to knowledge, Palaiseau, France.</td>
<td>grafematik2022.sciencesconf.org</td>
</tr>
<tr>
<td>Jun 8 – Jul 13</td>
<td>TypeParis22, intensive type design program, Paris, France.</td>
<td>typeparis.com</td>
</tr>
<tr>
<td>Jun 15</td>
<td><strong>TUG 2022</strong>, deadline for presentation proposals</td>
<td></td>
</tr>
<tr>
<td>Jun 23 – 25</td>
<td>DANTE 2022 Sommertagung and 64th meeting (hybrid), Otto-von-Guericke Universität, Magdeburg, Germany.</td>
<td>dante.de/veranstaltungen/dante2022</td>
</tr>
<tr>
<td>Jul 10</td>
<td><strong>TUG 2022</strong>, deadline for preprints for program</td>
<td></td>
</tr>
<tr>
<td>Jul 22 – 24</td>
<td>The 43rd annual meeting of the TeX Users Group.</td>
<td>tug.org/tug2022</td>
</tr>
<tr>
<td>Jul 31</td>
<td><strong>TUGboat</strong> 43:2 (proceedings issue), submission deadline.</td>
<td></td>
</tr>
<tr>
<td>Aug 8 – 11</td>
<td>SIGGRAPH 2022, Vancouver, Canada.</td>
<td>s2022.siggraph.org</td>
</tr>
<tr>
<td>Sep 12 – 18</td>
<td>16th International ConTEXt Meeting, Dreifelden, Germany.</td>
<td>meeting.contextgarden.net/2022</td>
</tr>
<tr>
<td>Oct 15</td>
<td><strong>TUGboat</strong> 42:3, submission deadline.</td>
<td></td>
</tr>
</tbody>
</table>

**Owing to the COVID-19 pandemic, schedules may change. Check the websites for details.**

**Status as of 15 April 2022**

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

7 Jacques André, Patrick Bideault, Denis Bitouzé, Michel Bovani, Thierry Bouche, Maxime Chupin, Daniel Filipo, Yvon Henel / The last decade at GUTenberg
  • response to article by Just; background and activities of GUTenberg

4 Barbara Beeton / Editorial comments
  • typography and TUGboat news

6 Jonathan Fine / Robin Fairbairns and UK TUG
  • personal remembrance of Robin

3 Boris Veytsman / From the president
  • on the use of inventions for war and peace

Intermediate

23 Seth Bergmann / Making open source textbooks, and diagrams with AlDraTex
  • collaborative textbooks and graphics with [Al]DraTex.sty

75 Karl Berry / The treasure chest
  • new CTAN packages, October 2021–April 2022

10 Vít Novotný, Dominik Rehák, Michal Hoftich, Tereza Vrabcová / Markdown 2.15.0: What’s new?
  • new features both for Markdown writers and coders of templates and solutions

Intermediate Plus

16 Carla Maggi / The DuckBoat — Beginners' Pond: CDs, but not Compact Disks
  • making commutative diagrams with the tikz-cd package

40 Joseph Wright / l3build: The beginner’s guide
  • testing, preparing, installing, and uploading releases of packages

Advanced

28 Max Chernoff / Automatically removing widows and orphans with lua-widow-control
  • automated removal of widow and orphan lines, without stretching, in all Lua formats

44 Nicola Talbot / bib2gls: standalone entries and repeated lists (a little book of poisons)
  • glossary definitions in documents; reordering glossary entries in multiple ways

59 Udo Wermuth / Transparent file I/O using the original \TeX program and the plain \TeX format
  • require user confirmation on \input and other operations

Reports and notices

2 Institutional members

73 Sarah Lang / Book reviews: \LaTeX Beginner’s Guide, second edition, by Stefan Kottwitz
  • review of this introduction for today’s new \LaTeX users

77 Karl Berry / \TeX Live 2022 news
  • brief summary of known issues in the TL’22 release

78 From other \TeX journals: Die \TeXnische Komödie 4/2021–1/2022; Zpravodaj 2021/1–4

80 TUG Bylaws committee / TUG Annual General Meeting Procedures

81 Karl Berry / TUG financial statements for 2021

82 \TeX consulting and production services

83 \TeXnology Inc.

84 Calendar