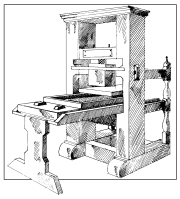


TUGboat

Volume 45, Number 3 / 2024

TEX Users Group

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

Individual TUG memberships

2025 dues for individual members are as follows:

- Trial rate for new members: $35.
- Regular members: $115.
- Special rate: $85.

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

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership carries with it such rights and responsibilities as voting in TUG elections. All the details are on the TUG web site.

Journal subscriptions

TUGboat subscriptions (non-voting) are available to 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 2025 is $130.

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 tug.org/instmem or contact the TUG office.

About the cover

The cover graphic was created by Keith McKay; it's an example of the art of asemic writing — writing without meaning. Keith's article on pages 340–345 gives some history and describes how it was made, using MetaFun and ConTEXt.

Trademarks

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

[printing date: October 2024]

Printed in U.S.A.

Board of Directors

Donald Knuth, *Ur Wizard of TEX-arcana*†
Arthur Rosendahl, *President**
Boris Veytsman*, *Vice President*
Karl Berry*, *Treasurer*
Jim Hefferon*, *Secretary*
Barbara Beeton*
Johannes Braams
Max Chernoff
Kaja Christiansen
Ulrike Fischer
Klaus Höppner
Tom Hejda
Jérémy Just
Frank Mittelbach
Ross Moore
Norbert Preining
Raymond Goucher (1937–2019), *Founding Executive Director*
Hermann Zapf (1918–2015), *Wizard of Fonts*

**member of executive committee*
†*honorary*

See 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

tug.org
tug.org/TUGboat

Electronic mail

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

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

Volunteer TEXnical support, public mailing list (a.k.a. texhax):
support@tug.org

Contact the Board of Directors:
board@tug.org

Copyright © 2024 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. An information notice to the *TUGboat* editors regarding such redistribution is appreciated.

A figure may have multiple parts. Each part of a figure should be labeled logically and unambiguously.

Mary Letourneau and Jennifer Wright Sharp
AMS *Style Guide: Journals* (October 2017)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
Editor Barbara Beeton

Volume 45, Number 3, 2024
Portland, Oregon, U.S.A.

TUGboat editorial information
This regular issue (Vol. 45, No. 3) is the last issue of the 2024 volume year. The deadline for the first issue of next year is March 21, 2025. 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 Editor*
Sophia 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. As of fall 2023, submission also implies permission to be indexed in the EBSCO (a large subscription management company) databases. We made this agreement with EBSCO to provide more visibility to *TUGboat* articles. See `tug.org/TUGboat/tubperm.html` for more.

If you have any concerns about these permissions, please notify the editors at the time of submission and we will do our best 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`
Thanks to all for their support!

American Mathematical Society, *Providence*, *Rhode Island.* `ams.org`

Association for Computing Machinery, *New York City*, *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.* `dukeupress.edu`

Hindawi Foundation, *London*, *UK.* `hindawi.org`

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

L3Harris, *Melbourne*, *Florida.* `l3harris.com`

LATEX 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`

Modular Font Editor K. `mfek.org`

Nagwa Limited, *Windsor*, *UK.* `nagwa.com`

NASA. `nasa.gov`

National Security Agency. `nsa.gov`

Ontario Tech University, *Oshawa*, *Ontario*, *Canada.* `ontariotechu.ca`

Overleaf, *London*, *UK.* `overleaf.com`

StackExchange, *New York City*, *New York.* `tex.stackexchange.com`

Tailor Swift Bot, *College Station*, *Texas.* `tailorswiftbot.com`

TEXFolio, *Trivandrum*, *India.* `texfolio.org`

Université Laval, *Ste-Foy*, *Québec*, *Canada.* `bibl.ulaval.ca`

University of Oslo, Institute of Informatics, *Blindern*, *Oslo*, *Norway.* `uio.no`

VTEX UAB, *Vilnius*, *Lithuania.* `vtex.lt`

2025 TEX Users Group election

TUG Elections committee

The terms of TUG President and six TUG Directors will expire as of the 2025 Annual Meeting, to be held in July 2025. One position is open; thus seven are to be filled.

The terms of these directors will expire in 2025: Karl Berry, Johannes Braams, Kaja Christiansen, Klaus Höppner, Frank Mittelbach, Ross Moore.

Continuing directors, with terms ending in 2027: Barbara Beeton, Max Chernoff, Ulrike Fischer, Jim Hefferon, Tom Hejda, Jérémy Just, Norbert Preining, Boris Veytsman.

The election to choose the new President and Directors will be held in early Spring of 2025. Nominations for these openings are now invited. A nomination form is available on this page or via `tug.org/election`.

The TUG Bylaws provide that "Any member may be nominated for election to the office of TUG President/ to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election ... shall be by ... ballot of the entire membership, carried out in accordance with those same Procedures."

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office; the petition and all signatures must be received by the deadline stated below. A candidate's membership dues for 2025 must be paid before the nomination deadline. The term of TUG President is two years, and the term of Director is four years.

A list of informal guidelines for all TUG board members is available at `tug.org/election/guidelines.html`. It describes the basic functioning of the TUG board, including roles for the various offices and ethical considerations. The expectation is that all board members will abide by the spirit of these guidelines.

Requirements for submitting a nomination are listed at the top of the form. The deadline for receipt of completed nomination forms and ballot information is

07:00 a.m. PST, Saturday, 1 March 2025

at the TUG office in Portland, Oregon, USA. No exceptions will be made. Forms may be submitted by fax, or scanned and submitted by email to `office@tug.org`; receipt will be confirmed by email. In case of any questions about a candidacy, the full TUG Board will be consulted.

Information for obtaining ballot forms from the TUG website will be distributed by email to all members within 21 days after the close of nominations. It will be possible to vote electronically. Members preferring to receive a paper ballot may make arrangements by notifying the TUG office; see address on the form. Marked ballots must be received by the date noted on the ballots.

Ballots will be counted by a disinterested party not affiliated with the TUG organization. The results of the election should be available by mid-April, and will be announced in a future issue of *TUGboat* and through various TEX-related electronic media.

2025 TUG Election — Nomination Form

Eligibility requirements:

- TUG members whose dues for 2025 have been paid.
- Signatures of two (2) members in good standing at the time they sign the nomination form.
- Supplementary material to be included with the form: passport-size photograph, a short biography, and a statement of intent. The biography and statement together may not exceed 400 words.
- Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

Name of Nominee: _____

Signature: _____

Date: _____

for the position of (check one):

☐ **TUG President**

☐ **Member of the TUG Board of Directors**

for a term beginning with the 2025 Annual Meeting.

1. _____ (please print)

   _____ (signature) _____ (date)

2. _____ (please print)

   _____ (signature) _____ (date)

Return this nomination form to the TUG office via postal mail, fax, or scanned and sent by email. Nomination forms and all required supplementary material (photograph, biography and personal statement for inclusion on the ballot, dues payment) must be received at the TUG office in Portland, Oregon, USA, no later than

07:00 a.m. PST, Saturday, 1 March 2025.

It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will late or incomplete applications be accepted.

Supplementary material may be sent separately from the form, and supporting signatures need not all appear on the same physical form.

☐ 2025 membership dues paid
☐ nomination form
☐ photograph
☐ biography/personal statement

TEX Users Group
Nominations for 2025 Election
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

(email: `office@tug.org`; fax: +1 815 301-3568)

doi.org/10.47397/tb/45-3/tb141elec

Editorial comments

Barbara Beeton

COVID persists

The usual lead article for a regular *TUGboat* issue is a message from the president. However, Arthur Rosendahl, TUG's current president, has contracted COVID, and isn't up to the effort for this issue.

We wish him a full and speedy recovery.

Alan Jeffrey, 1967–2024

Alan Jeffrey, an early member of the LaTeX Project Team, passed away on 4 July 2024, a victim of brain cancer. He was born on 17 January 1967 in Glasgow. At the tine of his death he was Principal Software Engineer at Roblox, Chicago, Illinois. His main interest was in the application of tools and techniques from CS research to practical problems.

The changes file for LaTeX 2$_\varepsilon$ shows entries by Alan from day one. A lot of his focus was on improving the font handling, including integration of NFSS. He was the original author of `fontinst`, and co-author of the `stmaryrd` Metafont, among other packages.[1] During the early 1990s, he contributed several articles to *TUGboat*, mainly about fonts.[2]

After his move to Chicago in the late 1990s, Alan had an active presence at `mastodon.scot/@asaj`. In it, he confides that he grew up in Glasgow. His later postings mostly chronicle the diagnosis of his illness, and its treatment. Honest, but sad.

This notice was posted online by his brother and sister, earlier this year:

> Alan's Celebration of Life is to be held at Wolfson College Oxford, where Alan completed his DPhil. The event will involve speeches and music, and will be live-streamed, to allow people to join from afar. There will be a buffet dinner, and, for those that wish to, we will go onwards to the pub.

The Celebration was scheduled for Saturday, November 2nd.

Alan's siblings also posted a more detailed remembrance on his LinkedIn account.[3]

R.I.P., Alan.

---

[1] `ctan.org/author/jeffrey`

[2] `tug.org/TUGboat/Contents/listauthor.html#Jeffrey,Alan`

[3] `www.linkedin.com/feed/update/urn:li:activity:7215033040614436865/`

Errata: *TUGboat* 7:4, "Amsterdam, 13 March 1996//Knuth meets NTG members", pp. 342–355

First of all, the "//" in the title should have been a double backslash, for a line break. Oops!

- Page 345, col. 1, paragraph 3, line 10: "Everytime" should be "Every time".
- Page 346, col. 1, paragraph 5, line 10: "be be able", only one "be".
- Page 343, col. 2, paragraph 3, line 3: "lionness" should be "lioness".
- Page 351, col. 1, line 4: "by saing" should be "by saying".
- Page 351, col. 2, paragraph 2, line 6: "*MetaPost*" should be the symbol ∓, caused by an unfortunate redefinition of the macro `\mp`.
- Page 353, col. 1, paragraph 3, line 5: "Dash" should be "`\Dash`", and should continue the previous line beginning with an em-dash.
- Page 354, col. 2, line 7: "gan influence" should be "gain influence".
- Page 354, col. 2, last paragraph, line 3: "the *The Art*", omit the lowercase roman "the".
- Page 355, col. 1, paragraph 2, lines 6–7: "TeX the Program", "METAFONT" and "[The] Stanford GraphBase" are the names of books and should be italicized.

Face/Interface conference at Stanford

This series of conferences, organized by Stanford's SILICON (Stanford Initiative on Language Inclusion and Conservation in Old and New Media) project.[4] will continue with a third edition, to be held from 13–18 January 2025. The date for submitting a proposal has already passed, but the posted call[5] holds a link to the application form, which in turn contains a profile of the keynote speaker, Adam Yeo, a font designer from Côte d'Ivoire who holds a Ph.D. in Art and Design from Nanjing University of the Arts, China, and is now Assistant Professor of Graphic Design at the University of Bondoukou, Ivory Coast.

The second edition of the conference, held in December 2023, was reported in an earlier column.[6] Recordings are still being processed, and will be posted on SILICON's new YouTube channel.[7] Chuck Bigelow's keynote address, "Prediction is Difficult, Especially About the Future", is already there, at `www.youtube.com/watch?v=tIO-ZgkPf-U`.

[4] `silicon.stanford.edu`

[5] `silicon.stanford.edu/face-interface/`

[6] `tug.org/TUGboat/tb45-1/tb139beet.pdf`

[7] `www.youtube.com/@StanfordSILICON`

Barbara Beeton

doi.org/10.47397/tb/45-3/tb141beet

A font walk in Rochester

Signs are everywhere; we depend on them to help us get to where we want to go. Sometimes they are easy to read, sometimes not, but when they're not, their intended function can be seriously compromised.

On an evening in July, Chuck Bigelow was walking with a neighbor, and he observed that the font on a street sign was less than optimally presented. One wouldn't be surprised that someone with Chuck's font experience would notice such flaws. What *is* surprising is that the neighbor wrote about the experience in the local newspaper, the *Rochester Beacon*, so others can benefit from Chuck's observations.

The article,[8] along with the font analysis, includes a biographical sketch and appropriately illustrative photos as well as a mention of Chuck's article "Oh, oh, zero!",[9] published here in *TUGboat* in 2013. Both articles are worth (re)reading.

At the end of the *Beacon* article, do continue reading the comments for additional typeface commentary.

Postscript. Other "font walks" can be cited. I've been told by Frank Romano (now the president of the Printing Museum in Haverhill, Massachusetts, but formerly on the faculty of the Rochester Institute of Technology at the same time that Hermann Zapf held the Cary Professorship in Graphic Arts) that he and Zapf took a similar walk through Rochester.

Don Knuth, to celebrate the end of a METAFONT course in 1984, took his students on a field trip to San Francisco, where they "had a picnic on Font Boulevard, then toured the fascinating MacKenzie-Harris type foundry and the Bigelow & Holmes design studio." They capped their tour by forming an "italic font" at the base of an appropriate street sign.[10]

And finally, as I was reminded by Lance Carnes after he had read the *Beacon* article, at a TUG conference at Stanford in the 1980s, I spotted and reported that "Santa Teresa St" on a sign near the cafeteria was spelled differently on the signs at other intersections. This is more a proofreading than a font observation, but demonstrates that reading street signs can provide more information than just where you happen to be located.

[8] rochesterbeacon.com/2024/07/29/seeing-rochester-signs-through-chuck-bigelows-eyes

[9] tug.org/TUGboat/tb34-2/tb107bigelow-zero.pdf

[10] See the photo on page 109 of the report, at tug.org/TUGboat/tb05-2/tb10knut.pdf.

Accessibility and arXiv

From September 3–13, arXiv presented Accessibility 2024, its second accessibility forum. In all, seven sessions were held online, organized around topics such as HTML papers on arXiv, the applicability of AI, and the deaf community at the Rochester Institute of Technology. One session was directed to the Spanish speaking community. All sessions have now been posted to YouTube.[11] Captions are present for all sessions, but some of them may still require editing; this effort is underway.

The session likely of most interest to TUG readers is the session on the HTML papers, in which Norbert Preining was a presenter.

What editor do you use for (LA)TEX input?

Earlier this fall, a number of "public" TEXies received a message with the subject "Research on LaTeX" asking these questions:

1. What editor do you use for work and why? Is it a free version or you pay for it?
2. What functionality do you use/like the most? And what do you lack?
3. What type of work do you use LaTeX for? For example, writing articles for scientific journals, research papers, etc.

The research in question is being carried out by a student at the State Technical University of the Kyrgyz Republic, Alex Surikov. His inquiry ended with the explanation "The problem is that I cannot find enough information to complete my research and would really appreciate your help." He included a list of the text editors that had already been reported. Especially since the list did not include the editor I use, I was happy to answer his questions, and invited him to report his results in *TUGboat*.

Since identifying LATEX users and finding out how to get in touch with them is not trivial, I suspect that Alex's respondent base is not as diverse as the general population of LATEX users. Therefore, I'm suggesting that if you read this, and haven't already sent Alex a response, you consider doing so. He can be reached at surikov_70@internet.ru.

◇ Barbara Beeton
https://tug.org/TUGboat

[11] www.youtube.com/playlist?list=PLYgeAMJvRZ6ZRuNQGoekx0FdjXqEG0bzM

Usenet and the future of `comp.text.tex`

Rayner Lucas, Tristan Miller, Marco Moock

Abstract

Usenet has served as a venue for announcements and discussions on the TEX ecosystem for over 40 years, and remains in operation despite the recent shuttering of the Web-to-news gateway on Google Groups. This article (re)introduces readers to Usenet and its `comp.text.tex` newsgroup, discusses the implications of their severance from Google, and explains how users can continue to access them.

1 Introduction

TUGboat recently printed an obituary of sorts for the `comp.text.tex` newsgroup [3], a demise it attributes to Google's decision to stop accepting posts from Usenet [9]. The present authors, all members of Usenet's Big-8 Management Board, would like to assure readers that `comp.text.tex` is still alive; the group remains available for anyone who wants to ask a question or talk about anything to do with TEX. In this article, we introduce Usenet and `comp.text.tex` for those who may not be familiar with them or aware of their history. We also explain what has happened to `comp.text.tex` and describe how to access the group now that Google's Web-to-news gateway is effectively shuttered.

2 The history of Usenet and `comp.text.tex`

Usenet is a system of forums for discussions and announcements, known as *newsgroups* or simply *groups*, covering almost every imaginable topic. It is similar in purpose to Web forums, mailing lists, or Facebook pages. But unlike these other venues, Usenet has no central server and no central administrators.[1] Instead, it is a worldwide network of servers that forward messages to each other until every server has a copy of each message. (See Figure 1.) These servers might be run by individuals, companies, schools, universities, or other organizations. Users generally have an account on one or more of the servers, and log in to read and post messages. This decentralized nature means that if one Usenet server shuts down, Usenet as a whole still exists. In contrast, if Facebook were to shut down, all its groups would be gone forever.

Usenet was established in 1980 and was initially distinct from the Internet, with separate servers

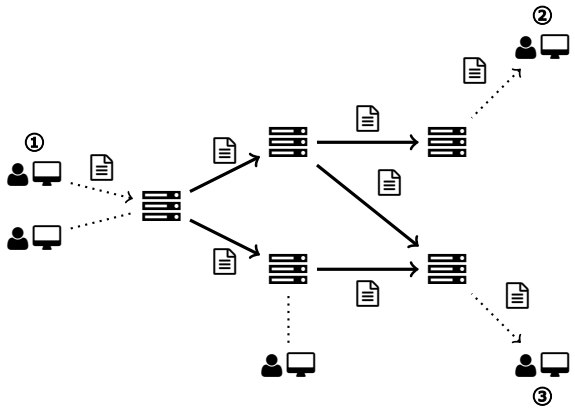


Figure 1: Usenet is a federated network of servers. A user (①) connects to a server and posts a message which then gets propagated to every other server in the network. Interested users (②, ③) can then download and read the message from the servers they use.

and communication protocols. However, links were quickly established via e-mail gateways and mirrors, and by 1986 it became fully integrated into the Internet with its adoption of the Network News Transfer Protocol (NNTP) [7]. By the 1990s, commercial Internet service providers were offering Usenet access as a standard service alongside e-mail and the World Wide Web. With the rise of the latter, a number of "Web-to-news" gateways sprang up allowing users to read and post to Usenet directly from their Web browsers. Perhaps the most popular of these gateways was Google Groups, which we cover in the next section.

Groups on Usenet are arranged into hierarchies according to topic. (See Figure 2.) Some top-level hierarchies are global and are carried by most servers — these include the Big 8 (`comp.*`, `humanities.*`, `misc.*`, `news.*`, `rec.*`, `sci.*`, `soc.*`, and `talk.*`) and the more freely managed `alt.*`. Other hierarchies, such as `europa.*` (Europe), `japan.*` (Japan), and `tor.*` (Toronto), are region- or language-specific and may be carried by only a subset of servers. Still other hierarchies might be carried by only a single server, such as the `panix.*` hierarchy, which contains technical support groups for customers of the Panix ISP.

Groups can be found at any level of the hierarchy except for the top level. For example, the top-level `comp.*` hierarchy contains groups related to computing. The `comp.text` group is for computer document formats, and contains a number of subgroups, including `comp.text.xml`, `comp.text.pdf`, and of course `comp.text.tex`, which is for discussions and announcements of anything related to TEX and friends.

[1] Despite our lofty-sounding name, the Usenet Big-8 Management Board neither claims nor exerts authority over Usenet. Rather, our role is to facilitate the community and technical processes for the creation and removal of newsgroups in eight of the network's oldest and largest hierarchies.

doi.org/10.47397/tb/45-3/tb141lucas-usenet

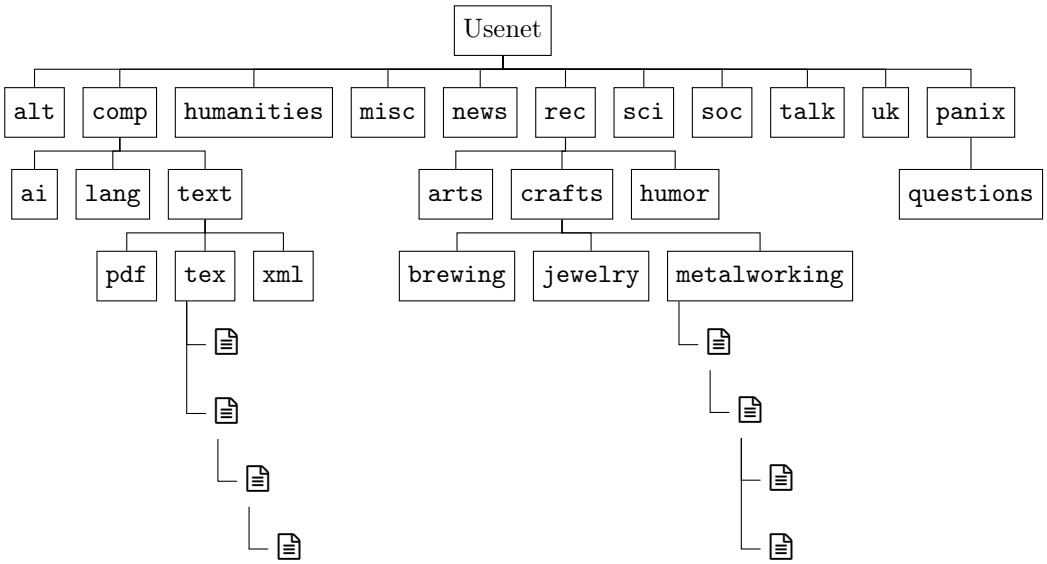


Figure 2: Structure of Usenet newsgroups, showing some of the global, regional, and local hierarchies and some of their individual groups, as well as the threaded posts in two of the groups.

TEX users and Usenet have had a very long association. `comp.text` was created in 1983 (initially as `net.text`, before the Great Renaming of 1987 [11]) and soon attracted conversations about TEX. (The earliest example we can find is a March 1984 post by Fritz Benedict of the University of Texas, who made the admittedly unusual request for TEX drivers for the Diablo 630 ECS daisy-wheel printer [4].) `comp.text.tex` was created in February 1990 at the instigation of Utrecht University's Piet van Oostrum, who gave the following rationale in his proposal and call for supporting votes [10]:

> The newsgroup is for TEX and LATEX related postings. These are now mainly found in `comp.text`, with some Post[S]cript dvi-driver related postings in `comp.lang.postscript`. About half (or a little bit more) of `comp.text` is devoted to TEX and LATEX, and a lot of TEXers rather would not be bothered by nroff/troff or even WordPerfect. This motivates the place in the hierarchy. [...]
>
> The creation of the newsgroup will also make it easier for the TEXhax Digest told [*sic*] be bi-directionally gatewayed. It is now gatewayed *into* `comp.text`, but for the benefit of those poor people without Usenet access, the other way would also be preferable.

The new group did not escape the notice of *TUGboat*, with which it shared many readers and contributors. Perhaps its first mention here is in a 1991 article by then–TUG president Nelson Beebe [2]; the supplement to the same issue gives a brief write-up of the group in a directory of TEX resources [1]. `comp.text.tex` quickly established itself as a major source of English-language news and support for the TEX ecosystem, alongside similar groups in region- or language-specific hierarchies such as `fr.comp.text.tex`, `de.comp.text.tex`, and `es.comp.lenguajes.tex`. It was the home of regular FAQ postings by Bobby Bodenheimer that were later adapted into the (UK) TEX FAQ [12], and its messages are cited in the source code or documentation of dozens of (LA)TEX packages.

3 Google Groups and its Web-to-news gateway

Throughout Usenet's history, a number of one- or two-way gateways have existed allowing users to read and/or post to newsgroups via e-mail, the Web, or other network services. The largest and most popular of the two-way Web-based gateways was Google Groups (`groups.google.com`), which became operational in 2001 following Google's acquisition of the Deja News service. Google Groups made it comparatively easy to access Usenet, and to search a sizeable archive of posts going back to 1981, through a user-friendly browser-based interface. To its credit, this significantly lowered the barrier to access Usenet's vibrant communities and high-quality information, particularly for users who first got online after the dominance of the Web in the mid- to late 1990s. However, Google also earned the ire of many established

Usenet users. It did relatively little to curb spam and other abusive posts originating from its servers, and failed to inculcate in its users the social conventions and niceties that Usenet had built up over the preceding decades. This latter problem was exacerbated by Google Groups's failure to distinguish between its own local forums and the ones gatewayed to Usenet.

Regardless of whether one views Google Groups as a net positive or net negative for Usenet, the fact remains that it became one of the most common ways for users to browse and post to newsgroups, including `comp.text.tex`. As with many of its other services, however, Google gradually let Google Groups fall into disrepair [5, 6, 8], and in 2023, it finally announced that it would stop updating its archive of Usenet groups in February 2024. While historical Usenet content remains visible there, it is not particularly easy to find or to search, and new Usenet posts are being accepted neither from local users nor from peering NNTP servers. Thanks to Usenet's federated nature, `comp.text.tex` and other newsgroups still exist separately from Google Groups, but Google users must now access them through some other method.

4 Connecting to Usenet today

Would-be visitors to `comp.text.tex` and other newsgroups still have a variety of options for gaining access. These include the following:

From the Web. Perhaps the simplest way to access Usenet is with a Web browser. Google Groups was the best-known way of doing this, but there are other websites that provide a similar free service. One such gateway is Newsgrouper (`cmacleod.me.uk/ng`): first follow the prompts to register and log in to your account, or else click the "Continue as Guest" button for read-only access; then enter a keyword or the name of a group (such as `comp.text.tex`) in the search box. Another gateway is Rocksolid BBS (`www.rocksolidbbs.com`): this one presents a list of all available newsgroups on a single page, which you can either scroll through or search using your browser's "Find" function. As with Newsgrouper, read access is available for all users; for posting access, first use the "register" link in the site's header. Other uni- or bidirectional Web-to-news gateways can be found on a list that we maintain at `www.big-8.org/wiki/Web-to-news_gateways`.

With an e-mail app. Some e-mail apps can also access Usenet groups over NNTP. Thunderbird is perhaps the best-known e-mail program that includes support for Usenet. You will first need an account with a Usenet provider. If you access the Internet through a university, large company, or commercial ISP, it may still run a Usenet server of its own that you can use — check your support documentation to see if this is the case and, if so, to get the connection details. If not, a free and well-regarded service you can use is Eternal September (`www.eternal-september.org`). You can then add your Usenet account to Thunderbird, in much the same way you would add a new e-mail account. We provide a step-by-step tutorial on how to do this at `www.big-8.org/wiki/Getting_Started_with_Usenet`.

With a newsreader. There are also dedicated apps, called *news clients* or *newsreaders*, for accessing Usenet. These generally look and work similarly to e-mail software such as Thunderbird, but have useful features specific to reading and posting to newsgroups. As above, you will need to sign up for an account with a Usenet provider first. We maintain a list of newsreader software, including both dedicated Usenet software and dual-purpose mail-and-Usenet apps, at `www.big-8.org/wiki/Newsreaders`.

5 Other concerns

Those contemplating joining Usenet, or returning to it after a long absence, may harbour certain apprehensions concerning its content and quality. In this section, we address some commonly expressed concerns.

Spam. As a fairly open network, Usenet has (or at least had) a not entirely undeserved reputation for hosting large amounts of spam. Indeed, anyone who used Google Groups will have noticed that many groups there were filled with off-topic commercial solicitations. However, as discussed above, Google Groups had poor spam filtering, and much of the junk was being posted from Google accounts in the first place. Today's reputable Usenet services have highly effective spam filters, and providers often cooperate to identify and remove spam promptly. This means that users of these servers see comparatively little of the spam that gets posted to Usenet nowadays (and also little of the spam that was historically posted through Google Groups). Most e-mail and newsreader apps will also let you set up your own filters: this means you can filter out any spam that hasn't already been caught by your provider, and also means you can ignore any topics or individuals that you don't want to see.

Usenet as a file-sharing service. Usenet has newsgroups for uploading files, often called *binaries* to distinguish them from text-based discussion. These groups have become extremely popular and some commercial Usenet services market themselves

chiefly towards file-sharers. This has created a misconception among certain Internet users that Usenet is nothing more than a shady file-sharing service. However, Usenet has always been a discussion platform at its core, and many of its users are there to participate in communities based around common interests. Free and non-commercial Usenet providers usually do not offer access to the file-sharing groups, as they are much more costly in storage and network capacity than simple text-based groups.

Active and inactive newsgroups. In the days before social media and the Web, many lively communities formed on Usenet. Since then, some newsgroups have fallen into disuse as users have drifted away to other venues, or the group's topic has become obsolete. However, there are still numerous groups with daily activity and regular users. One of them is `comp.text.tex` itself, which gets several new messages on an average day. Here's a selection of subject lines from the group's most recent posts, excluding CTAN announcements:

- `manually installing tex package groups in Ubuntu`
- `pgfplots: link to top of embedded page`
- `S equation numbering`
- `Long left right harpoons`
- `[LaTeX][PGF/TikZ] Undefined control sequence error for PGF intersections.`
- `Combination with Hoefler Text`
- `[LaTeX] How to tell the user that the document wasn't compiled with -shell-escape flag?`
- `TeX's line breaking in the grub sesh`
- `"Force" location of pictures.`

We keep a list of active groups at `www.big-8.org/w/images/3/3e/Sample-newsrc.txt`; this list is in a standard, text-based format that can be imported as a subscription list by many newsreaders.

6 Conclusion

In short, Usenet and `comp.text.tex` still exist, and new participants are welcome. Please join the group to ask a question, share your own knowledge, or just browse through some of the previous discussions and announcements. We hope to see you there.

References

[1] `comp.text.tex`. *TUGboat* 12(2):171 (supplement), June 1991.

[2] N.H.F. Beebe. President's introduction. *TUGboat* 12(2):205–208, June 1991. `tug.org/TUGboat/tb12-2/tb32pres.pdf`

[3] B. Beeton. Editorial comments. *TUGboat* 45(1):4–6, 2024. `doi.org/10.47397/tb/45-1/tb139beet`

[4] F. Benedict. Wanted: Diablo 630 ECS driver for TEX. `net.text`, `<87@utastro.UUCP>`, 27 Mar. 1984.

[5] M. Braga. Google, a search company, has made its Internet archive impossible to search. *Motherboard*, 13 Feb. 2015. `www.vice.com/en/article/jp5a77/google-a-search-company-has-made-its-internet-archive-impossible-to-search`

[6] J. Corbet. Historical programming-language groups disappearing from Google. *LWN.net*, 28 July 2020. `lwn.net/Articles/827233/`

[7] B. Kantor, P. Lapsley. Network News Transfer Protocol. RFC 977, Feb. 1986. `doi.org/10.17487/RFC0977`

[8] C. Paloque-Bergès. Usenet as a Web archive: Multi-layered archives of computer-mediated-communication. In *Web 25: Histories from the first 25 Years of the World Wide Web*, N. Brügger, ed., Digital Formations. Peter Lang, 2017. `halshs.archives-ouvertes.fr/halshs-01843627`

[9] R. Pegoraro. End of an era: Google Groups to drop Usenet support. *PCMag*, 16 Dec. 2023. `www.pcmag.com/news/end-of-an-era-google-groups-to-drop-usenet-support`

[10] P. van Oostrum. Call for votes: `comp.text.tex`. `news.announce.newgroups`, `<2293@ruuinf.cs.ruu.nl>`, 14 Jan. 1990.

[11] Wikipedia. Great Renaming. `en.wikipedia.org/wiki/Great_Renaming`

[12] J. Wright. Development of the UK TEX FAQ. *TUGboat* 36(2):105–105, 2015. `tug.org/TUGboat/tb36-2/tb113wright-faq.pdf`

◇ Rayner Lucas
Usenet Big-8 Management Board
`rlucas (at) big-8 dot org`
`https://www.big-8.org/`

◇ Tristan Miller
Usenet Big-8 Management Board
`tmiller (at) big-8 dot org`
`https://www.big-8.org/`
ORCID 0000-0002-0749-1100

◇ Marco Moock
Usenet Big-8 Management Board
`mmoock (at) big-8 dot org`
`https://www.big-8.org/`

Typographers' Inn

Peter Flynn

TUG logo

A logotype was originally a word in a font cast as a single piece of type, for example *The* in ATF Garamond, or (now digitally speaking) TEX [2, p. 266]. The shortened form 'logo' now refers to any design or symbol, typographic or not, which is used to identify a concept, organization, or product.

Figure 1: The TUG logo from 1998

The TEX Users Group has been using the logo in Figure 1 since 1998, although its origins were in the early 1990s. It uses Optima and Garamond, which for TEX users at the time meant buying the Type 1 fonts and creating `.tfm` and `.vf` files.

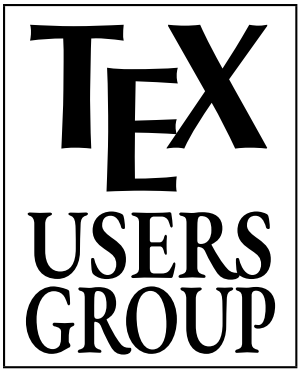


Figure 2: Partial reconstruction of the TUG logo using fonts from the LATEX Font Catalogue

It's easier to create nowadays, with XELATEX and QT Optimum and EB Garamond OpenType fonts as not-quite-perfect substitutes, as in Figure 2. I have not implemented the broken rule and the drop shadow; the spacing in the TEX logo itself needs adjusting; and it probably needs Adobe Garamond Pro Bold with the shorter swash foot to the R (this one uses the bold from EB Garamond, so the long foot of the R had to be accommodated with some manual kerning).

Figure 3: Early tests on a modified TUG logo using free fonts

The point is that you can get very close to faking it up with existing resources in LATEX, which would mean it could be implemented as a package, and therefore generated as part of the normal PDF output, rather than inserted as a JPG image.

But do we need to? The question has been asked before: indeed Karl Berry had to remind me that there was a competition in 2009 for a new logo for TUG (see `tug.org/logo/competition/` for submissions), but none of the submissions was selected. More recently, Karl and others tried various free fonts to test the feasibility in the existing vertical boxed layout (Figure 3). Tests were also done on a wider, shallower horizontal layout (Figure 4).

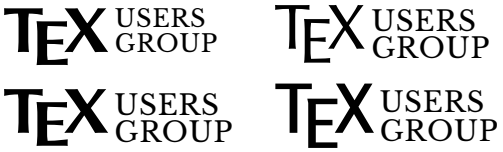


Figure 4: Early tests on a horizontal layout for the TUG logo using free fonts

However, much has changed since 2009, so is it time to hold another competition for a new logo? And maybe not one based on the existing design at all, but still carrying the message that TEX is a typesetting system.

There would need to be a brief, of course, for the guidance of designers, which might include some, more, or none of the following:

- it has to be done using TEX in one of its flavours;
- it must use only features that exist in the standard TEX Live distribution;
- it must use fonts which are either in the TEX Live distribution or can be downloaded from CTAN;
- it has to contain the words 'TEX', 'users', and 'group', preferably in that order;

doi.org/10.47397/tb/45-3/tb141inn

- it should provide a square or nearly-square version, so that it can conveniently be thumbnailed for use as avatars, contacts lists, and webpage favicons;
- Brownie points if the logo can also be approximated in plain text so that it can be used in places with restricted formatting facilities (for example, Figure 5).

```
+-------+
|  TeX  |
| USERS |
| GROUP |
+-------+
```

Figure 5: TEX logo approximated in plain text

Otherwise I don't think there should be any restrictions on how it looks, what colors (if any) are used, or fonts, or anything else about layout. Designer briefs that are too tight are almost worse than those which are too loose.

(To digress; In the early 1970s I went to a series of early evening concerts in London's South Bank for which the programme (UK spelling) was photocopied on A3 folded once to A4 — but had been prepared entirely on an IBM Selectric 'golfball' typewriter, so just the one font (Courier). But the spacing and layout was so cleverly done that I suspect most people never noticed it had not been typeset. If you're a good enough designer, I suspect you can work even with the most limited resources.[1])

Figure 6: TUG logo revisited

Maybe this is something that should be discussed in the usual forums like `comp.text.tex`, `tex.stackexchange.com`, Discord, and the mailing lists or forums of all the TEX users groups worldwide.

I couldn't design my way out of a paper bag, but consider Figure 6 my contribution.

[1] I'm sure I have written about this before, but I cannot find the reference.

'Sitting by Nellie'

This phrase was the nickname for that type of training where you sat beside an experienced operator and learned from them.[2] I mentioned her in a paper at the TUG meeting in 2005 [1], and she got another outing in the 'Ask Nelly' column of the PracTEX Journal [3].

If Nellie was a skilled operator *and* good at imparting information *and* patient with novices and slow learners, *and* if the newcomer was willing to learn, they would probably learn fast and become productive. If not ... well, we have probably all seen the typographic effects of learning (LA)TEX by sitting beside a friend or colleague in the office, laboratory, library, computer room, or wherever. The learner picks up the rudiments, but also picks up all the bad habits or [now-]obsolete practices which Nellie herself acquired (quite blamelessly) when learning (LA)TEX the same way a generation or more before.

Over the past couple of years, I have been lurking on the group chat system Discord, which was originally set up for gamers, but which now has channels ('servers' in Discord-speak) on all topics, including TEX. While there are some deep and serious discussions about TEX internals and primitives (most of which is beyond me), the majority of questions I have seen so far are about LATEX output not looking 'right': generally errors caused by mistakes they don't understand, and simple unawareness of the nature and capabilities of LATEX but also specific typographic features 'not working', as listed below.

I have answered some of the questions which I felt were within my range of competence. In almost every case this has revealed that the poster is a newcomer, often a student with a project deadline, who has picked up some rudiments of LATEX from Nellie — in reality a friend or colleague or teacher or even a web page — but crucially has unknowingly been impeded or misled by a Google search without a knowledge of the terms which might elicit a meaningful answer. The clue is often some phrase like 'I've been told that ...'.

Leaving aside the mathematical, structural, and syntactic questions, most of the remaining stumbling-blocks in the typography of a document seem to be caused by a lack of conceptual information:

- a belief that `\em`, `\bf`, `\it`, `\sl`, `\sc`, and `\tt` take an argument;

[2] I first came across Nellie in an industrial context, but her origins may be military — in the post-WWII period there was a large influx of ex-service people into industry, bringing with them much of the military ethos which businesses hoped would improve performance.

- non-Latin–1 characters not rendering because they are still using *pdflatex* with inputenc and fontenc instead of *xelatex* with fontspec;[3]
- unawareness of floats, and their connection to page size and shape limits;
- attempts to 'fix' [old] *bibtex* problems instead of using biblatex and *biber*;
- uncritical enthusiasm for 'solutions' generated by AI systems at the expense of RTFM;
- use of some *Overleaf* 'templates' which are undocumented downloads from GitHub;
- unawareness of the existence of the log file, so font substitutions may go unchecked.

In every single case I looked at, the workable solution was well documented, but as one user put it, 'only findable by reading the documentation, which is only readable by finding it.' Similarly, the right keywords for a web search are only learned with experience.

In many cases, reading and re-reading and reading the question again reveals that the user can see the problem but does not have the words in their vocabulary to express it. The result is that 'title' and 'heading' get conflated, as do 'line' and 'rule', 'block' and 'column' (or 'figure' or 'table'), and the distinctions between font and typeface, roman and bold and italics, and (strangely) size. And I'm only scratching the surface here.

So what can we do to improve users' typesetting quality, especially for new users who are migrating from Word? Training is not a viable option — at least, not in any sense that we can manage it, and we are ruling out Nellie *ex hypothesi*. Documentation or self-study doesn't seem to be an answer, as they don't read it except when pressured to do so. Web pages addressing users' specific problems often exist in forums like `tex.stackexchange.com`, and in old posts to `comp.text.tex`, but finding them is becoming increasingly difficult as the search-engine æther becomes more polluted by AI. And one-on-one Q&A like Discord is unsustainable, even if it is fun watching people fix things themselves.

Answers on a postcard, please.

Afterthought

Research into editors for LaTeX is rare, so when Aleksei Surikov from the State Technical University of the Kyrgyz Republic contacted me with details of ongoing research in the area I felt it was something worth a wider audience. He is looking into features like the availability of templates, real-time collaboration; file sharing, mathematical symbols, automatic recognition of warnings and errors, spell-checking, syntax highlighting, reference checking and others. Some of these ('templates' and math symbols) are available in all editors, of course, but many of the other features vary from system to system.

Specifically, he's trying to identify usage:

- what editor people use, and why;
- whether is it cost-free or commercial;
- what the user's favorite functionality is, and what is lacking;
- what type of work it's used for (articles, handouts, notes, papers, books, etc.).

You may contact him directly at алексей суриков at `surikov_70@internet.ru`.

All of which is strongly reminiscent of the questions Word users ask about LaTeX, as to whether or not it could handle this format or that layout or this font or that style, to which the response was that it had everything Word had, as shown in Figure 7.

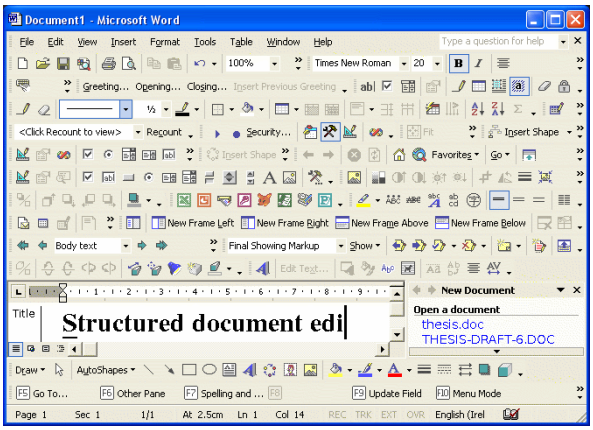


Figure 7: Trying to use Microsoft Word for your thesis

References

[1] P. Flynn. TeX and the Interfaces — Practical TeX 2004 keynote address. *TUGboat* 25(1):31–34, 2004. `tug.org/TUGboat/tb25-1/flynn.pdf`

[2] P. Flynn. Typographers' Inn. *TUGboat*, 37(3):264–266, Apr. 2016. `tug.org/TUGboat/tb37-3/tb117inn.pdf`

[3] PracTeX Editors. Ask Nelly. *PracTeX Journal*, 1, Jan. 2005. `tug.org/pracjourn/2005-1/asknelly/`

◇ Peter Flynn
Textual Therapy Division,
Silmaril Consultants
Cork, Ireland
Phone: +353 86 824 5333
`peter (at) silmaril dot ie`
`blogs.silmaril.ie/peter`

[3] All the test files users showed or uploaded *were* UTF-8, so the old problem of using elderly encodings seems to have passed.

LaTeX in the Digital Humanities

Sarah Lang

Abstract

This paper explores the intersection of LaTeX typesetting and the digital humanities. More specifically, it asks why LaTeX typesetting is used so infrequently in the Digital Humanities (DH), despite its clear advantages and applications. This paper, on the one hand, aims to understand the reasons why LaTeX isn't more widely used in the digital humanities and, on the other hand, presents three examples of relevant use cases to illustrate its value for the (Digital) Humanities.

1 What are the Digital Humanities?

Despite having become established as a field in recent decades, defining Digital Humanities (DH) remains difficult because the field encompasses a range of methods, tools, and subject materials. It integrates both digital and computational approaches with Humanities research as an interdisciplinary field.[1] Broadly speaking, Digital Humanities refers to the use of digital methods to analyze Humanities data and to address Humanities research questions. Roth [13] has proposed distinguishing three subfields:

1. **Humanities of the Digital:** This branch focuses on media studies and similar matters.[2]
2. **Digitized Humanities:** As suggested by the name, this means hermeneutically traditional humanities work (such as text editing, annotation, or interpretation) but moved to digital spaces and technologies such as TEI-XML (to be discussed later).
3. **Computational Humanities:** Here, advanced computational techniques — such as machine learning, large-scale text analysis, or computer vision — are applied to Humanities data. While these methods offer powerful new approaches, they often come with unexpected challenges in applying them to humanities data. This often proves surprisingly difficult and the methods follow epistemologies fundamentally different from traditional Humanities work.

2 Use cases for LaTeX in DH

1. **Using LaTeX in conference submissions** is a more universal approach to the use of LaTeX, primarily related to the presentation and publication of research rather than the content itself. Its benefits lie in how effectively it handles complex documents, making it a useful tool across various fields, including the humanities.
2. **For archaeological and other catalogues or large scholarly monographs** in the humanities, LaTeX is very well suited. Monographs replete with images tend to be unwieldy in software like MS Word, whereas LaTeX handles these with ease and isn't that hard to learn, even for humanists.[3]
3. **In digital scholarly editions**, producing print versions is probably one of the most interesting uses of LaTeX for Digital Humanists [10, 11].

We'll discuss each of these in more detail.

2.1 Use case 1: Conference proceedings in LaTeX

Conference proceedings that require full paper submissions, such as those for by the *Computational Humanities Research* Conference present a good reason to get acquainted with LaTeX.[4] However, the use of LaTeX has been criticized in some Digital Humanities (DH) circles as exclusionary, as noted by Dombrowski [1]. While I agree that not all DH work requires coding, I don't see why LaTeX is particularly singled out, especially since it is not as difficult to learn compared to other tools increasingly common in DH. The sentiment of alienation seems less about the technology itself and more tied to the association of LaTeX with Computer Science, a field not known for pioneering inclusivity and diversity.

In my view, there are several advantages to using LaTeX for conference submissions: It allows the outsourcing of typesetting work to authors via standardized LaTeX templates. This makes sense for smaller conferences run by a community, though it does place some additional burden on editors and organizers. Humanities scholars, even those with limited technical expertise, can learn LaTeX as a markup language to format their submissions effectively. Many are already familiar with markup languages, making LaTeX far easier to grasp than programming languages like Python, which are becoming more prevalent in DH.[5] Additionally, LaTeX facilitates fast, open-access publication of conference proceedings, which benefits early-career scholars and provides an alternative to the exploitative practices often seen in commercial academic publishing.

[1] Some even argue that field doesn't want to define itself [12]. My definitions follow [6], [12], and [13].

[2] This is not my flavour of DH and seems to me especially dominant in the Anglophone area.

[3] See an example of typesetting archaeological catalogues in [7].

[4] `computational-humanities-research.org`

[5] See tutorials specifically for Humanists in [4, 8].

doi.org/10.47397/tb/45-3/tb141lang-dh

2.2 Use case 2: Printed books and archaeological catalogues

2.2.1 Physical books in the Humanities

Despite early predictions that digital media would render physical books obsolete, by 2024 it is evident that this shift has not occurred and seems unlikely to happen in the near future. The value of the book as a material object and a symbol of cultural capital has retained its significance. It has even gained renewed symbolic value among younger audiences on platforms like BookTok or celebrities who hire book curators for their homes and to decide which books they should be photographed holding. This suggests a renewed appreciation of books not only as material objects but also as symbols of cultural capital. Moreover, many scholars in the Humanities continue to prefer printed materials for tasks such as close reading, underscoring the book's enduring role in academic practice.

2.2.2 LaTeX in archaeology

In disciplines such as archaeology, compiling extensive catalogs of objects or findings is a common component of larger research endeavors, such as PhD dissertations. These catalogs often contain numerous images and, due to their size, can become difficult to manage using standard word processing tools like MS Word. While this usage extends beyond the Digital Humanities, it highlights the cross-disciplinary relevance of LaTeX.

As previously discussed, LaTeX excels in typesetting large scholarly monographs and managing extensive catalogs. Its capacity to handle large datasets (e.g., with tools like `csvsimple`, see [7]) and incorporate numerous images efficiently makes it an ideal tool for managing complex archaeological projects.

2.3 Use case 3: Print versions for digital scholarly editions

Perhaps the most significant use case for LaTeX within the Digital Humanities is digital scholarly editing (DSE [14]). DSE is a core task in the DH community. This is particularly true in contexts such as my institution in Graz, where DSE traditionally holds a key position, even as the subfield of Computational Humanities gains momentum. This leads me to assume that digital scholarly editing will likely remain a core technology within Digital Humanities.

Given this, and as physical books and print editions persist and show no signs of disappearing, LaTeX remains relevant for the production and dissemination of scholarly editions and research outputs.

2.3.1 Transforming TEI-XML to LaTeX

The Text Encoding Initiative (TEI) is the standard XML format used for archiving and managing data in digital scholarly editions.

Transforming digital scholarly editions in TEI-XML format to LaTeX enables the creation of high-quality print versions from digital editions, even when the primary focus of the edition is not on print publication. Using XSLT to transform TEI-XML data into LaTeX facilitates the generation of print-ready material with minimal effort. This process could potentially be enhanced by the use of large language models to automate the generation of XSLT code (or similar), further streamlining the workflow for scholars in the future. This caters to the continuing demand from traditional humanists for print(-able) editions. These print versions are particularly beneficial in educational settings where physical materials remain a preferred medium for study and instruction.

The `reledmac` package [16] is especially valuable for Digital Humanists to ensure the accessibility and usability of scholarly resources in both digital and print formats, as it supports complex typesetting requirements often encountered in scholarly editions [2].

2.3.2 TEI primer

The following is a minimal example of TEI-XML:

```
<TEI> <!-- root element -->
<teiHeader>
<!-- author, title, dating,
        sources, edition rules, etc. -->
</teiHeader>
<text> ... </text>
</TEI>
```

TEI further offers elements such as `<div>` (division) for a text division, `<hi>` (highlight), `<pb>` (page beginning) and a range of specialized modules. Thus, it allows for the encoding of complex textual information, such as multiple versions of a text or critical apparatus for documenting textual variants across different witnesses in a stemma.

In TEI, a critical apparatus is encoded using several elements to capture textual variants across witnesses. The `<app>` (apparatus entry) containing one entry in a critical apparatus,[6] with `<rdg>` (reading) and `<lem>` (lemma) tags capturing the actual variations. Multiple readings can be grouped within a `<rdgGrp>` if necessary.

Notably, `<lem>` can contain text that does not appear in any of the textual witnesses, allowing for

[6] `tei-c.org/release/doc/tei-p5-doc/en/html/TC.html#TCAPLL`

editorial interventions or reconstructions of the text in the base text. The following shows an example:

```
<app>
  <rdgGrp type="orthographic">
    <rdg wit="#Sh1">giue</rdg>
    <lem wit="#Sh2">give</rdg>
  </rdgGrp>
  <rdg wit="#AS1">have</rdg>
</app>
<listWit>
  <witness xml:id="Sh1">
    <bibl>Folger STC 22276</bibl>
  </witness>
  <witness xml:id="Sh2">
    <bibl>Huntington 69304</bibl>
  </witness>
</listWit>
```

2.3.3 Typesetting critical editions with the `reledmac` package

A critical apparatus documenting textual variants found in different witnesses of a text's transmission requires specialized tools for typesetting; the `reledmac` package is designed to manage such complex editorial needs. TEI encodes text semantically using XML, but to produce a print version, it needs to be transformed into the presentational markup of LaTeX.

The TEI Apparatus Toolbox (TEI-CAT), developed by Marjorie Burghart, offers a ready-made solution for this transformation.[7] It allows scholars to print an edition of a TEI-XML document by converting it into LaTeX using the `reledmac` package and subsequently compile it to PDF. This transformation is achieved via an XSLT script, customizable through a web interface. The tool offers a practical way to automate the typesetting of TEI-encoded critical editions, although it does have limitations in terms of fine-tuning the final layout and appearance.[8]

The XSLT template used for transforming TEI into LaTeX using `reledmac` is quite complicated, as it has to account for various encoding possibilities within TEI and integrates the customization options from the web form. However, this is hard to follow for experts in XSLT and far too difficult for beginners. Thus, I developed simplified XSLT templates for teaching purposes, aimed at helping learners understand the basics of transforming TEI into LaTeX [5].[9]

3 Outlook

3.1 Popularizing LaTeX in Digital Humanities

While LaTeX has significant potential in the Digital Humanities (DH), its adoption remains limited. In DH, many projects prioritize digital outputs over printed formats. Despite the clear advantages of LaTeX for typesetting and managing large, complex documents, it remains underutilized in the field.

However, XSLT-based transformations offer a practical way to generate high-quality print versions from TEI-encoded digital editions, a crucial capability as the alternative XSL-FO (Formatting Objects) standard was discontinued in 2012. Additionally, the rise of large language models presents new possibilities for automating and simplifying these transformations, making the creation of print-ready editions from TEI data even more accessible.

Print outputs remain valuable, particularly for textual scholars who continue to favor working with printed materials for tasks such as close reading and annotation. By addressing these opportunities, LaTeX could become more integral to the DH toolkit.

3.2 Practical implementation challenges in the GAMS repository

In the course of working on a digital scholarly edition project, I encountered several practical challenges with using LaTeX for generating print outputs intended for teaching materials [3].

Our editions are archived in the Humanities Asset Management System (GAMS) [15], which holds XML data from which it dynamically generates various representations such as HTML, print or RDF, following the single-source principle. This included creating PDFs from LaTeX through a multi-step process. The archived XML data used to be transformed into LaTeX code using XSLT via the `getLaTeXPDF()` method, and the LaTeX was then compiled into PDFs

[7] `teicat.huma-num.fr/print.php`

[8] For more details on the XSLT used in this process, see the XSL file, available here: `github.com/MarjorieBurghart/TEI-CAT/blob/master/tei2latex_final.xslt`. The transformation is triggered by a web form through which limited customization is possible.

[9] These simplified XSLT documents were a key component of the first (and, thus far, last) in-person LaTeX Ninja workshop, held in 2022 at Harvard University [9].

The workshop provided participants with hands-on experience in using these templates to convert TEI into LaTeX, making the learning curve for beginners more manageable while still covering essential concepts of the transformation process. This included a simplified version of the TEI-CAT template used to generate `reledmac` LaTeX from TEI-XML, which is available here: `github.com/sarahalang/Harvard_BeyondTEI_Workshop_SLang2022/blob/main/ADDITIONAL_RESOURCES/XSL_BASE_STYLESHEETS/mini-latex-reledmac.xsl`

using Python's Rubber.[10] However, this process encountered several issues.

In the case of the transformation to LaTeX, the underlying LaTeX code was not always visible, making errors cryptic and time-consuming to resolve. Compounding this, even documents that had been successfully compiled in the past sometimes produced errors in subsequent attempts, adding unpredictability to the process. To troubleshoot, I often had to run the XSLT stylesheet locally, contrary to the intended automatic transformation on the system. This step added extra work, as it became necessary to manually verify the outputs.

A subsequent "dockerization" of GAMS complicated matters further, as some LaTeX functionalities were not carried over due to their limited use. This affected the `getLaTeX()` function that used to show the LaTeX code generated through the XSLT transformation. Without the ability to inspect the generated LaTeX code before compilation, debugging became even more difficult. Errors often stemmed from the XSLT transformation and resulted from discrepancies between XML and LaTeX, such as nesting and whitespace handling. Despite efforts to include BibTeX in the LaTeX file using `filecontents`, the system failed to compile it properly. The container used for compiling LaTeX was sizeable and retained outdated auxiliary files, often requiring manual intervention.

Ultimately, dynamic LaTeX generation from XML proved overly complex, resource-intensive, and prone to errors. Most users needed static outputs rather than dynamic ones, making it more efficient to compile LaTeX locally and archive static PDFs. This was already evident in the aforementioned GRaF project, where attempts to dynamically generate LaTeX outputs for teaching materials proved cumbersome for both stylesheet developers and system administrators. Ultimately, the team opted for manual compilation and static PDF archiving as a simpler, more reliable solution and discontinued the dynamic compilation from XML.[11] Moving forward, it will be necessary to compile LaTeX locally and upload static outputs archived in the repository.

Still, XML-to-LaTeX transformations may remain useful tools for complex print outputs, such as critical editions, in the digital scholarly editing community, despite the fact that these use cases were rare in the Graz GAMS infrastructure and thus, difficult to maintain.

[10] `pypi.org/project/latex-rubber/`

[11] A final documentation of the removed features can be found here: `github.com/sarahalang/latex-for-dh/blob/main/latex-on-gams.md`

3.3 Future directions

A tutorial and starter XSLT documents could significantly lower the barrier to entry for new users, providing a practical foundation for those unfamiliar with these workflows. I am considering creating a tutorial on the *Programming Historian*, teaching LaTeX basics and TEI-to-LaTeX via XSLT transformations in an accessible way, as well as explaining their value for digital humanists.[12] This step-by-step guide would demonstrate the process of converting TEI-encoded texts into LaTeX using XSLT and walk users through the creation of print-ready PDFs from XML-encoded digital editions.

[12] `programminghistorian.org`

References

[1] Q. Dombrowski. Does coding matter for doing digital humanities? In *The Bloomsbury Handbook to the Digital Humanities*, J. O'Sullivan, ed., Bloomsbury Handbooks, pp. 137–146, London, 2022. Bloomsbury Academic. `doi.org/10.5040/9781350232143.ch-13`

[2] A.N.J. Dunning. Reledmac. Typesetting technology-independent critical editions with LaTeX. *RIDE – A review journal for digital editions and resources*, 11, 2020. Tools and Environments. `ride.i-d-e.de/issues/issue-11/reledmac/`

[3] Grazer Repositorium antiker Fabeln (GRaF). Ursula Gärtner (ed.): `gams.uni-graz.at/graf`, 2020.

[4] S. Lang. A Humanities seminar paper with LaTeX - in 10 minutes. The LaTeX Ninja Blog, Jan. 2019. `latex-ninja.com/2019/01/22/a-humanities-seminar-paper-with-latex-in-10-minutes/`

[5] S. Lang. Simple XML to LaTeX transformation tutorial. The LaTeX Ninja Blog, Feb. 2019. `latex-ninja.com/2019/02/18/xml-to-latex-simple/`

[6] S. Lang. Experiments in the digital laboratory. What the Computational Humanities can learn about their definition and terminology from the History of Science. In *Fabrikation von Erkenntnis: Experimente in den Digital Humanities*, M. Burghardt, L. Dieckmann, et al., eds., Esch-sur-Alzette, 2021. Melusina Press. `doi.org/10.26298/melusina.8f8w-y749-eitd`

[7] S. Lang. LaTeX for archaeologists: An archaeological catalogue from a spreadsheet. The LaTeX Ninja Blog, July 2021. `latex-ninja.com/2021/07/16/latex-for-archaeologists-an-archaeological-catalogue-from-a-spreadsheet/`

[8] S. Lang. The LaTeX newbie's guide to using Overleaf for conference paper submissions.

The LaTeX Ninja Blog, Jan. 2022. latex-ninja.com/2022/01/09/the-latex-newbies-guide-to-using-overleaf-for-conference-paper-submissions/

[9] S. Lang. First ever LaTeX Ninja workshop at Harvard: Beyond TEI: Digital editions with XPath and XSLT for the web and in LaTeX. The LaTeX Ninja Blog, Apr. 2022. latex-ninja.com/2022/04/17/first-ever-latex-ninja-workshop-at-harvard-beyond-tei-digital-editions-with-xpath-and-xslt-for-the-web-and-in-latex/

[10] S. Lang. Enough reledmac to be dangerous: Scholarly editing with LaTeX & XSLT. The LaTeX Ninja Blog, Sept. 2022. latex-ninja.com/2022/09/03/enough-reledmac-to-be-dangerous-scholarly-editing-with-latex-xslt/

[11] S. Lang. What you really need to know about digital scholarly editing. The LaTeX Ninja Blog, Oct. 2022. latex-ninja.com/2022/10/30/what-you-really-need-to-know-about-digital-scholarly-editing/

[12] M. Piotrowski. Ain't no way around it: Why we need to be clear about what we mean by "digital humanities". In *Wozu Digitale Geisteswissenschaften? Innovationen, Revisionen, Binnenkonflikte (November 20–22, 2019)*, Lüneburg, 2019. Leuphana University Lüneburg.

[13] C. Roth. Digital, digitized, and numerical humanities. *Digital Scholarship in the Humanities*, 34/3:616–632, Sept. 2019. doi.org/10.1093/llc/fqy057

[14] P. Sahle. What is a scholarly digital edition? In *Digital Scholarly Editing: Theories and Practices*, M.J. Driscoll, E. Pierazzo, eds., pp. 19–39. Open Book Publisher, Cambridge, 2016. books.openedition.org/obp/3381

[15] J. Stigler, E. Steiner. GAMS – eine Infrastruktur zur Langzeitarchivierung und Publikation geisteswissenschaftlicher Forschungsdaten. *Mitteilungen der Vereinigung österreichischer Bibliothekarinnen und Bibliothekare*, 71(1):207–216, 2018. doi.org/10.31263/voebm.v71i1.1992

[16] P.R. Wilson, M. Rouquette. *reledmac — Typeset scholarly editions*. Comprehensive TEX Archive Network (CTAN), ctan.org/pkg/reledmac, May 2024.

◇ Sarah Lang
Elisabethstraße 59
8042 Graz
Austria
sarah dot lang (at) uni-graz dot at
https://sarahalang.com
ORCID 0000-0002-4618-9481

Enhancing TEX hyphenation rules for Portuguese

Leonardo Araujo, Aline Benevides

Abstract

Portuguese hyphenation rules have been available for more than 35 years and have performed well. Nonetheless, they still make mistakes and leave some hyphenation points unmarked. Although most undetected hyphenation points are located near word boundaries, which are irrelevant for TEX typesetting purposes, they are still useful for hyphenating proper nouns, new words, or pseudowords, and for use in other applications, such as text-to-speech conversion. A list of 49,528 hyphenated words acquired from online dictionaries was used to analyze the default rules' hyphenation performance, and a set of 120 new rules is proposed to be added to the hyphenation rules for Portuguese. In addition, Patgen was used to create new rules from scratch or to improve the previous set of rules. Patgen's set of rules doesn't show good generalization capability. In the end, the set of handcrafted patched rules shows the best results.

1 Introduction

Automatic hyphenation is a crucial part of document preparation systems, especially in typesetting and desktop publishing. It improves readability and aesthetics in text layout. It avoids the necessity of editors or typographers intervening to insert hyphens at suitable locations and it can be adjusted to different languages.

Despite the fact that the TEX hyphenation algorithm and rules are old, they are, to the present, the most frequently used approach, even outside the TEX world. The grounds for this is Hunspell, a spell checker and morphological analyzer that is adopted in many software packages, e.g., LibreOffice, Mozilla Firefox, Mozilla Thunderbird, Google Chrome, macOS, InDesign, memoQ, Opera, Affinity Publisher, and others [10]. Hunspell uses TEX hyphenation rules [11, 13], making TEX hyphenation widespread in the computer world. That is a result of TEX's approach: simplicity and versatility. The algorithm works effectively, as it already supports rules for 66 languages [20], and offers the flexibility to create rules for any currently unsupported language.

In this paper, we evaluate the current hyphenation rules initially proposed by de Rezende (1987) [6] and later updated by de Rezende and Almeida (2015) [7]. We defined our gold standard hyphenation dictionary based on six different Portuguese online dictionaries, and used it to analyze the performance

doi.org/10.47397/tb/45-3/tb141araujo-pthyph

of different sets of rules. Using grammatical rules and phonological principles, we proposed a set of rules to complement the existing standard hyphenation rules used in TEX, and we also created a new set of rules using Patgen.

This paper is organized as follows: Section 2 discusses hyphenation as a general tool; Section 3 reviews the existing TEX hyphenation rules for Portuguese; Section 4 describes the creation of our hyphenation dictionary to be used as a reference; Section 5 presents the results of applying the default TEX rules to this dictionary; Section 6 summarizes our proposed handcrafted patch rules to address errors found in the default rules; in Section 7 we use Patgen to create a set of rules from scratch, built on the default rules or built on the here-proposed set of rules; finally, we conclude the paper in Section 8.

2 Hyphenation

Hyphenation in text wrapping was not used for a long time. Words should fit entirely in a line, or they would be broken in arbitrary places. Initially, no markers were used to indicate word wrapping, leading to potential confusion and unintended interpretations. As a result, orthographers advocated for the introduction of a sign to indicate such breaks. Portuguese faced the same gradual introduction of a hyphenation sign to mark words wrapping across lines. Even though the usage of a hyphenation sign (=) was advocated by orthographers [5], few documents used such a sign until the end of the 18th century [4].

In some cases, hyphenation can hinder smooth reading and should be avoided. For example, in children's literature, it can disrupt the reading flow and comprehension. Similarly, in large fonts or headlines, hyphenation appears visually unappealing. In technical documents, it may cause confusion when applied to specialized terms. In contrast, large or small spaces between words can also impose difficulty on the reading process, making hyphenation fundamental when texts use short line lengths. As a line gets shorter, the number of breaking candidates between words decreases, leading to awkward spaces between words and among letters. In left-aligned text, consistent spacing can eliminate the risk of river formation, ensuring a consistent and aesthetically pleasing appearance of the text. Likewise, in justified text, hyphenation can create awkward spacing, reducing readability. Therefore, automatic hyphenation plays an important role in good typesetting.

Another important matter to consider is ambiguities that might arise when a word is partitioned during the hyphenation process. In English, we should avoid hyphenations such as *re-cover*, *re-form*, *re-sign*, *the-rapist*, *depart-mental* and *mans-laughter* (for Portuguese, some examples are: *de-putada*, *fede-ração*, *acu-mula*, *após-tolo*, *cú-bico*). Hyphenations that might lead the reader to pronounce a word incorrectly should also be prevented (e.g., *consider-ation* in English, and *pe-rigo* in Portuguese). Even though a native English speaker may recognize the word *consideration*, an undesired hyphenation like *considera-tion* could mislead an inattentive reader into an incorrect pronunciation (/kənsɪdəɹə-tɪɑn/ instead of /kənsɪdəɹeɪ-ʃən/). The same argument goes for *perigo* (could be erroneously pronounced as /pe-xigʊ/ instead of /pe-ɾigʊ/).

In some situations, hyphenation is also a matter of style. Some partitioning choices sound better than others. These conflicting alternatives typically arise when a word has many possible hyphenation points. Consider *ar-chae-ol-o-gist* (or *ar-che-ol-o-gist*), which is preferably partitioned as *archae-ologist* (or *arche-ologist*) in opposition to *archaeol-ogist* (or *archeol-ogist*) or *archaeolo-gist* (or *archeolo-gist*). It is preferable to keep whole morphemes together. In the previous example: *archae* (or *arche*, meaning ancient, primitive) and *ologist* (one who studies the topic). In Portuguese, the word *en-tres-sa-fra* is preferably split as *entres-safra* in opposition to *en-tressafra* or *entressa-fra*, and the word *fe-liz-men-te* is preferably split as *feliz-mente* in opposition to *felizmente* or *felizmen-te*. In both cases, keeping the two morphemes together was the primary motivation. It is also more elegant to avoid splits between double consonants or vowels, even if a hyphenation point exists between those letters. For example, *pressurizar* is preferable to *pres-surizar* and *empreen-dedor* is preferable to *empre-endedor*. Even so, exceptions exist; it is preferable to partition *micro-organismo* rather than *microor-ganismo* (keeping morphemes together is preferable to splitting a double vowel).

Analysis of only the immediate context may not be enough to determine a potential hyphenation point. For example, consider *dem-o-crat* and *democ-ra-cy*, where the letters adjacent to the *e* do not suffice to determine the break location. Even having chosen patterns that makes hyphenation relatively straightforward, there might be exceptions. Consider the sequence *tion*. As it is commonly a suffix in English, for morphological reasons, it is preferable to always pause before this pattern, as in *celebra-tion*, *explana-tion* and *motiva-tion*. However, the word *cation* is hyphenated as *cat-ion*, with its etymology determining the way this word is split.

A word with different meanings may be hyphenated differently according to its meaning. "For instance, the Swedish word form *glassko* has three different meanings, and can be hyphenated as *glas-sko* (glass shoe), *glass-ko* (ice cream cow) and in the non-standard way, *glass-sko* (ice cream shoe)." [18] In Portuguese, the word *sublinha* might be hyphenated in two different forms: *su-bli-nha* when representing the inflected form of verb *sublinhar* (underline as a verb) or *sub-li-nha* when referring to the line under (underline as a noun).

The general rule for Portuguese hyphenation is to split a word into its syllables. A syllable is made of a mandatory nucleus, filled by a vowel, and optional peripheral consonants (before or after the nucleus). In some situations, the syllabic division does not respect the morphological constituents, creating a conflicting relation with the morphological preference described previously. Prefixes *bis-*, *des-* and *in-* are examples. The correct syllabifications are *bi-sa-vô*, *de-sem-bar-que*, *i-na-ti-vo* and *i-nobs-tan-te*,[1] where the prefixes are split into two syllables.

Each language has its hyphenation rules, which can be categorized into four groups: morphology (etymology), focusing on meaningful word parts (prefixes, suffixes); phonology, driven by syllable breaks in speech; orthography, based on standard spelling conventions; and semantics, which considers context to avoid ambiguous or awkward splits. An algorithmic approach employs a logical system to analyze words and apply the hyphenation rules of a specific language. Since hyphenation rules vary significantly between languages, an algorithm must be developed for each one. While a logic-based system can be efficient and compact, it still needs to address exceptions through hard-coded rules.

The automation of this process might use a dictionary-based approach, which will restrict the hyphenation possibilities to those entries in the dictionary, or a algorithmic-based approach, which might be applied to whatever sequence is found in a text. An algorithmic approach might use a logic system to analyse words and apply the hyphenation rules of a given language. A rule-based model might include the recognition of prefixes, suffixes, morphemes, or other sequences suitable for the inclusion of a break. Another approach is the use of pattern matching. By using a corpus of hyphenation examples in a language, this approach identifies letter sequences that determine suitable hyphenation points. Patterns encompass prefixes, suffixes, exceptions, and special hyphenation rules of the language. Finally, a mixed model might also be used, combining two or more of the previously described methods to enhance hyphenation accuracy and flexibility.

The original TEX hyphenation algorithm was introduced by Knuth (1977) [12]. It was focused on the English language and used four main rules: (1) each part should contain a vowel other than final *e*; (2) suffix removal; (3) prefix removal; and (4) vowel-consonant-consonant-vowel (VCCV) breaking (but combine *h* with the previous letter if it is a consonant). Tests showed this could find 40% of the allowed hyphen locations [15].

The hyphenation algorithm proposed by Franklin M. Liang and adopted in TEX uses the notion of competing patterns [15]. A database of hyphenated words is swept looking for hyphenating and inhibiting patterns. The algorithm introduced in TEX82 uses five alternating levels of hyphenating and inhibiting patterns. Patgen, the program for pattern generation based on a corpus, was created by Liang [14] and was used to create hyphenating patterns for many languages [23, 25, 26, 29, 27].

The effective hyphenation of words by TEX depends on the following factors: (1) document language, which will determine which set of patterns to apply; (2) characters used, since some might block hyphenation at their edges; (3) the value of the internal variables `\lefthyphenmin` and `\righthyphenmin`, which defines the minimum sequence length of characters at the left and right borders before any hyphenation is allowed [28].

3 TEX hyphenation rules for Portuguese

TEX hyphenation rules for Portuguese have 307 rules in total [6, 7], and correctly hyphenate 92% of Portuguese words when analyzing the hyphenation dictionary proposed here (see Section 4). By observing the erroneous or missing cases, we analyze here the default rules to identify areas for improvements.

Out of the default TEX rules, 252 (82%) follow the pattern `1CV`, which represents the recurring CV syllables in Portuguese. As for the written consonant characters, there are typically 18 considered (`b`, `c`, `ç`, `d`, `f`, `g`, `j`, `k`, `l`, `m`, `n`, `p`, `r`, `s`, `t`, `v`, `x`, and `z`), which can combine with 14 written vowel characters (`a`, `e`, `i`, `o`, `u`, `á`, `â`, `ã`, `é`, `í`, `ó`, `ú`, `ê`, and `õ`), resulting in these CV patterns that indicate favorable hyphenation points before the consonants.[2] It is worth noting that there

[1] The rule of syllable division could lead to two possible partitions, based on phonological or morphological forces: *i-nobs-tan-te* and *in-obs-tan-te*, but the first is preferable.

[2] In Portuguese, usually the letter *h* appears at word initial position or after the consonants *c*, *l*, or *n*. Therefore, it is not worth encouraging a hyphenation point before an *h*V. Examples of words that have an *h* in the middle and not preceded by *c*, *l*, or *n* are: *Bahia*, *Corinthians*, *show*, *shopping*,

might be other rules or exceptions in the hyphenation patterns not covered by these default rules. To accommodate exceptions, the following rules were proposed [6]:

- 20 rules created for cases involving consonants `b`, `c`, `d`, `f`, `g`, `k`, `p`, `t`, `v`, or `w`[3] followed by `l` or `r`;
- 3 rules for `c`, `l` or `n` followed by `h`;
- 23 distinct patterns were introduced to indicate hyphenation points between vowels or between `c`'s, `r`'s, and `s`'s;
- 8 patterns adhere to the `1[gq]u4V` pattern, signaling beneficial hyphenation points before `g` or `q`, followed by a sequence of `u` and a vowel, with an inhibiting point between the `u` and the subsequent vowel;
- 1 pattern written as `1-`, denoting that a hyphen indeed serves as a beneficial hyphenation point.

4 Creating the reference Portuguese hyphenation dictionary

Our set of Portuguese words was created from those in the CETENFolha corpus [16], augmented with the word list from Palavras NET [19] and from the Portuguese Wikipedia dump [32]. From the Wikipedia data, we narrowed down the selection to a subset of 50,721 words, accounting for 95% of occurrences in the Wikipedia dump (the initial set had 419,578 words). This threshold was set to filter out typos and infrequent words. Finally, we refined the list by retaining only those words for which we could find hyphenation data in at least one of the following online dictionaries: Michaelis [17], Priberam [22], Wikcionário [31], Aulete [1], Portal da Língua Portuguesa [21], and Dicio [8]. This curation process resulted in a final dictionary containing 85,638 words. We also manually performed corrections in 10 entries that we judged necessary.

The fact that the Portuguese language relies on phonology as a key factor for hyphenation requires an understanding of how stress influences this process. Every word in the language with two or more syllables has one more-prominent syllable, which is considered the stressed syllable. The stress can fall on any of the last three syllables of the word, counted from the right end. Words are classified as oxytone when the last syllable is stressed, paroxytone when the penultimate syllable is stressed, or proparoxytone when the third-to-last syllable is stressed. The stress pattern considered unmarked (the most common) does not receive a diacritical mark; the marked patterns, including all proparoxytones, do receive a diacritical mark. This marking serves to indicate to the reader that the stressed syllable follows a less common pattern. The use of diacritical marks to indicate stress is an important clue for determining how a word can be hyphenated. For example, the words *saúde* and *saída*, which have stress marked on the vowels *u* and *i*, respectively, clearly indicate that these vowels form their own syllables: *sa-ú-de* and *sa-í-da*. However, there are ambiguous cases, both for speakers and in linguistic literature, such as *his-tória* (or *his-tó-ri-a*) and *car-tó-rio* (or *car-tó-ri-o*). The question is whether final vowel clusters like *-ia* and *-io* are diphthongs or hiatuses. Normative grammar indicates they are paroxytones, but also acknowledges that they can be apparent proparoxytones [2, 3].

In the Portuguese Wiktionary we found 1,848 words marked as apparent proparoxytones; these are words that appear to have stress on the antepenultimate syllable due to post-tonic vowel sequences treated as rising diphthongs. Despite their appearance, they follow the accentuation rules for paroxytones. Apparent proparoxytones can create confusion in syllabification and hyphenation because their stress pattern suggests different breaking points. For example, the word *história* could be hyphenated in two ways: *his-tó-ri-a* (proparoxytone) or *his-tó-ria* (paroxytone). This dual potential arises from treating the final vowel sequences as either separate syllables or as parts of diphthongs. In linguistic terms, accurate syllabification should consider the intended pronunciation and morphological structure, aligning with standard accentuation rules for the language [24]. For those words marked as apparent proparoxytones, we have accepted both hyphenations.

We have extracted 4 sub-dictionaries from the created hyphenation dictionary: the first dictionary is made of 15,842 words where the hyphenation given is the same in all six dictionaries; the next dictionary is made of 15,642 where five out of the six dictionaries agree on the hyphenation; then we have a dictionary with 10,299 words with four agreed hyphenations, and finally a dictionary with 7,745 words where just three dictionaries agree on the hyphenation.

Those four dictionaries were used to gradually incorporate new rules to fix hyphenation issues in Portuguese, whether by analyzing and proposing fixes or by using Patgen to create a new set of rules.

Matheus, and *sushi*. They are proper names or loanwords whose spellings have not been adapted to Portuguese.

[3] We don't know why they added the rules `1w2l` and `1w2r`, since there is no Portuguese word with such a sequence; we could find only foreignisms like *showroom*, *wrestling*, *bowling*, or other rather rare foreign words.

5 Performance of the default rules

In this section, we evaluate the effectiveness of the default TEX hyphenation rules for Portuguese by testing them against the comprehensive hyphenation dictionaries described in Section 4. This analysis helps us understand the accuracy and limitations of the existing rules and sets a baseline for further enhancements. The results of this evaluation are presented in Table 1.

Table 1: Results of TEX default hyphenation rules on the dictionaries created from online dictionaries hyphenations.

word list	correct	incorrect	missing	entries
6 agree	98.1%	0.2%	1.7%	15,842
5 agree	89.4%	7.3%	3.8%	15,642
4 agree	87.4%	8.6%	5.0%	10,299
3 agree	93.9%	1.0%	5.4%	7,766
total	92.5%	4.3%	3.7%	49,549

Some words might have both a bad and a missing hyphenation point, therefore the percentage in each line might add up to more than 100%.

The results show that the default TEX hyphenation rules perform quite well, with an overall accuracy of 92.5% across all word lists. The highest accuracy is observed in the '6 agree' category, with 98.1% correct hyphenations, indicating that the rules share a high correspondence with dictionaries on those words with strong consensus among them. However, there is a notable increase in incorrect and missing hyphenations as the consensus among sources decreases.

These findings highlight the robustness of the default rules in many cases but also reveal their limitations, particularly when dealing with less frequently hyphenated words. When regarding the frequency of occurrences of words in the Wikipedia corpus, the words with hyphenation errors amount to 3.7% of the occurrences in the corpus.

While the default TEX hyphenation rules demonstrate a high level of accuracy, there remains room for improvement. The next sections will explore ways to enhance these rules, including the development of handcrafted patch rules and the use of Patgen to create a new set of rules from scratch. These efforts aim to further reduce hyphenation errors and improve the overall performance of the TEX hyphenation system for Portuguese.

6 Handcrafted rules

We systematize the set of new rules in this section, and provide a few examples for each instance. These rules are intended as a complement for the default TEX hyphenation rules created by [7].

1 `.g2no`, `.g2nó`, `.g2nô` – *gnomo*, *gnóstico*, *gnômon*

2 `t2c` – *tchau*, *tcheco*

3 `1p2neu` – *pneumonia*, *pneumotórax*, *pneumático*, *hidropneumático*

4 `.t2m` – *tmese*

5 `.p2t` – *ptose*, *pterossauro*

6 `.m2n` – *mnemônico*

7 `c2za` – *czar*

8 `.s2` – *stalinismo*

9 `.t2` – *tsunami*, *tzarista*

10 `.p2si`, `.p2sí` – *psicologia*, *psíquico*

11 `su2b3r`, `su2b3l` – *subrotina*, *sublunar*
exception: `.su3b4li` – *sublinhar*, *sublimar*

12 `.ne4o` – *neoliberal*, *neonazista*

13 `1p2seu1d` – *pseudônimo*

14 `1qu` – *enquanto*, *inquieto*, *farquhar*, *qubit*

15 `a1ir.`, `u1ir.` – *sair*, *extrair*, *diminuir*, *incluir*

16 `a1ind`, `a1i1nh` – *ainda*, *rainha*

17 `e1imp` – *reimpresso*, *teleimpressor*

18 `e1inc`, `e1inf`, `e1ing`, `e1ins`, `e1int`, `e1inv` – *reincidência*, *reinfecção*, *reingressa*, *reinserção*, *reintegração*, *reinventar*

19 `u1iz.`, `a1iz.` – *juiz*, *raiz*

20 `pro1i1b` – *proibição*

21 `tu1i`, `bu1i`, `nu1i`, `o1im`, `o1in`, `u1in`, `su1i`, `í1e`, `ju1i`, `fu1i`, `du1i`, `do1im`, `au1i`, `u1i1ç` – *intuitivo*, *contribuidor*, *ingenuidade*, *coimbra*, *coincide*, *ruindade*, *suicida*, *píer*, *juizado*, *fuinha*, *assiduidade*, *amendoim*, *cacauicultor*, *constituição*
exception: `tu2id`, `tu2it`, `co2ima`, `o2i1na` – *gratuidade*, *intuito*, *coima*, *boina*

22 `a1â`, `a1ã`, `a1é`, `a1í`, `a1ó`, `a1ô`, `a1ú`, `e1á`, `e1â`, `e1ã`, `e1é`, `e1ê`, `e1í`, `e1ó`, `e1ô`, `e1ú`, `é1o`, `i1á`, `i1ã`, `i1é`, `i1í`, `i1ó`, `i1u`, `i1ú`, `í1a`, `í1o`, `o1á`, `o1ã`, `o1é`, `o1ê`, `o1í`, `o1ó`, `u1á`, `u1ã`, `u1â`, `u1é`, `u1ê`, `u1í`, `ú1o` – *abraâmico*, *abraão*, *aéreo*, *país*, *caótico*, *faraônico*, *saúde*, *balneário*, *oceânico*, *campeã*, *feérico*, *preênsil*, *veículo*, *teórico*, *napoleônico*, *conteúdo*, *néon*, *diário*, *região*, *soviético*, *iídiche*, *periódico*, *feiura*, *viúva*, *maníaco*, *íon*, *razoável*, *joão*, *poético*, *boêmia*, *heroísmo*, *alcoólico*, *usuário*, *itapuã*, *lituânia*, *suécia*, *cauê*, *suíça*, *flúor*
exception: `1gu2á`, `1gu2ã`, `1gu2é`, `1gu2ê`, `1gu2í`, `1qu2á`, `1qu2ã`, `1qu2â`, `1qu2é`, `1qu2ê`, `1qu2í`, – *jaraguá*, *saguão*, *alguém*, *português*, *linguística*, *aquático*, *camaquã*, *equânime*, *inquérito*, *sequência*, *química*

23 `1bô`, `1cô`, `1çô`, `1dô`, `1fô`, `1gô`, `1lô`, `1mô`, `1nô`, `1pô`, `1rô`, `1sô`, `1tô`, `1vô`, `1xô`, `1zô` – *robô*, *recôncavo*, *maçônico*, *judô*, *telefônica*, *xangô*, *camelô*, *sumô*, *econômico*, *capô*, *tarô*, *subsônico*, *chatô*, *vovô*, *saxônia*, *amazônia*

24 `4a.`, `4e.`, `4o.` – *secretária*, *planície*, *paratormônio*

The 120 rules were grouped above in a list of 24 types of rules. They may be further organized into five large groups. The first, which comprises rules 1 to 9, includes consonant clusters such as *czar*, *ptose* and *gnomo*. The second group, comprising rules 10 to 13, delimits the morphological boundary between prefixes and radicals. As noted, although phonological issues guide the separation of numerous words in Portuguese, there are also those that are guided by morphology. This is the case of words that have the prefixes *sub-* and *re-*, such as *sublunar* and *reinserção*.

The third group, comprising rules 14 to 22, seeks to handle a set of words that have vowel combinations that do not follow the general rules. This is because the Portuguese language has vowel encounters with the second vowel graphically marked that can be separated, forming hiatuses, such as *caótico*, *balneário* and *razoável*, while there are also words with a similar structure that constitute a diphthong, such as *português*, *alguém* and *linguística*. It is notable that the latter examples are formed by the digraphs *qu-* and *gu-*, while the former involve vowels other than *i* and *u*. In Portuguese, a diphthong with an accented second vowel is only possible when preceded by *qu-* or *gu*.

The fourth group, in turn, which comprises rules 22 and 23,[4] which are counterparts of rules that were already in the default rules, but did not contemplate the cases with certain diacritics. They were then added to encompass words such as *camelô*, *recôncavo*, *amazônia*, and *maçônico*.

The fifth, and last group, has just a single instance, rule 24, which represents our choice in how to deal the apparent proparoxytones, avoiding a final hyphenation with the vowels *a*, *e*, or *o*.

The full set of 120 rules is as follows:

.p2si	.p2sí	.g2no	.g2nó	.g2nô
t2c	1p2neu	.t2m	.p2t	su2b3r
su2b3l	.su3b4li	a1ir.	u1ir.	1qu
1vô	1lô	1cô	1gô	1bô
1tô	1rô	1pô	a1é	a1í
a1ó	a1ú	e1á	e1â	e1ã
e1é	e1ê	e1í	e1ó	é1o
e1ú	i1á	i1ã	íla	i1é
i1í	i1ó	ílo	i1u	i1ú
o1á	o1é	o1í	o1ó	u1á
u1ã	u1â	u1í	úlo	1qu2á
1qu2â	1qu2í	1gu2í	a1ind	a1i1nh
e1imp	e1inc	e1inf	e1ing	e1ins
e1int	e1inv	u1iz.	a1iz.	4a.
4e.	4o.	1gu2á	1gu2ã	1qu2ã
.m2n	c2za	.s2	1p2seu1d	1dô
1fô	1mô	1nô	1sô	1zô
tu1i	tu2it	tu2id	bu1i	nu1i
o1in	u1in	su1i	íle	ju1i
fu1i	du1i	do1im	au1i	u1i1ç
u1ê	1gu2ê	1qu2ê	1çô	u1é
1gu2é	1qu2é	1xô	a1â	a1ã
a1ô	e1ô	.ne4o	o1ã	o1ê
o1im	o2i1na	pro1i1b	co2ima	.t2

Rule 24 was created to inhibit final hyphenations, addressing errors arising from apparent proparoxytones. However, this resulted in missing hyphenations before a final vowel, which is a minor issue since such hyphenations are typically avoided to prevent orphan vowels.[5]

Tables 2 and 3 present the results for our patched hyphenation rules and for the rules with rule 24 removed, respectively.

Considering the frequency of occurrences, the proportion of words with hyphenation errors has decreased from 3.7% to 2.5% in the Wikipedia corpus.

[4] Note that rule 22 is in the intersection between the third and fourth group of rules.

[5] Avoiding widows and orphans is also a goal of TEX.

Table 2: Results of patched TEX hyphenation rules on the dictionaries created from online dictionaries hyphenations.

word list	correct	incorrect	missing	entries
6 agree	96.6%	0.0%	3.4%	15,842
5 agree	94.7%	0.1%	5.2%	15,642
4 agree	94.1%	0.1%	5.9%	10,299
3 agree	90.6%	0.5%	9.1%	7,766
total	94.6%	0.1%	5.4%	49,549

Some words might have a bad and a missing hyphenation point concomitantly, therefore the percentage in each line might add up to more than 100%.

Table 3: Results of patched TEX hyphenation rules without three rules created to deal with apparent proparoxytones (rule 24).

word list	correct	incorrect	missing	entries
6 agree	99.98%	0.0%	0.03%	15,842
5 agree	92.7%	7.1%	0.3%	15,642
4 agree	91.2%	8.2%	0.7%	10,299
3 agree	98.2%	0.6%	1.4%	7,766
total	95.6%	4.0%	0.5%	49,549

Some words might have a bad and a missing hyphenation point concomitantly, therefore the percentage in each line might add up to more than 100%.

While this represents a slight improvement, further error reductions may require a significant increase in the number and complexity of the rules, highlighting the diminishing returns on such refinements.

7 Patgen rules

Patgen was used to create new sets of rules from scratch, as well as rules starting from the default ones or from the patched rules proposed in Section 6. This involves defining specific parameters, choosing a reference dictionary with correct hyphenations, and using a translation file tailored for Patgen.

There are infinite ways to specify the parameters for Patgen. Based on previous works, we opted to test two approaches: (1) using fixed parameters at all stages of Patgen execution (starting from an empty set or from the default rules) [9]; (2) using a progressive approach, starting from an empty set and creating higher-order rules at each step, always building on the rules of the previous step [30].

As a reference dictionary, we used the concatenation of the dictionaries with six, five, and four agreements. The last dictionary (three agreements) was used as a validation dictionary.[6]

For more information on how to use Patgen and create the translation file, we refer the reader to [9].

Table 4: Results of patched TEX hyphenation rules created by Patgen on the validation dictionary with 7,745 entries.

model	correct	incorrect	missing	rules
fix† null	93.94%	1.26%	5.23%	849
fix† default	94.34%	0.84%	5.04%	1,091
fix† patched	94.46%	0.81%	4.94%	1,045
mfp‡ null	8.36%	0.47%	91.56%	699
mfp‡ default	94.23%	0.66%	5.32%	1,230
mfp‡ patched	93.70%	1.00%	5.52%	1,230

Some words might have a bad and a missing hyphenation point concomitantly, therefore the percentage in each line might add up to more than 100%.
† fix null/default/patched refers to the method using fixed parameters with a given initial rules set.
‡ mfp null/default/patched refers to the incremental approach using a given set of rules as a starting point.

Patgen creates a large set of rules that perform well within those specific sets, but have poor generalization capacity. Using smaller reference dictionaries will decrease performance, but that is primarily a consequence of Patgen including long rules that resemble specific exceptions rather than general rules. Some examples include: `3linea`, `neári5` and `padri3`.

[6] We have also tested starting with a smaller reference dictionary (using those with six and five agreements) and validating on the subsequent dictionaries, but the results are worse and therefore not discussed here.

It is also unrealistic to try to find a linguistic explanation for all rules created by Patgen, especially those involving large patterns.

Comparing the results from Table 4 with those from Tables 2 and 3, it becomes evident that Patgen's performance falls short when evaluated using the same validation dictionary.

8 Conclusion

Upon evaluating the effectiveness of the default TEX hyphenation rules for Portuguese using our hyphenation dictionaries, several key insights emerged. The results, as shown in Table 1, highlight the performance metrics of the default rules and set the baseline for improvements. Our handcrafted rules, designed to address specific hyphenation errors, demonstrated superior performance and generalization capability, as shown in Tables 2 and 3. Patgen was utilized to generate new sets of rules, both from scratch and by building upon the default rules or our handcrafted rules. Despite the potential of Patgen, its performance was suboptimal when evaluated with the same validation dictionary, as observed in Table 4. This was primarily due to the creation of overly specific rules that failed to generalize well.

The comprehensive analysis reveals that while default and automatically generated rules have their merits, manually refined rules significantly enhance hyphenation accuracy. This enhancement is crucial for typesetting high-quality Portuguese texts, ensuring readability and maintaining aesthetic consistency. Although there remains potential for further refinement of the rules, it may not be worthwhile since our dataset already encompasses the majority of words in the Portuguese vocabulary (95% of all occurrences in Portuguese Wikipedia).

We believe that more advanced hyphenation could require new algorithms or modifications to TEX's behavior, allowing it to account for other driving forces in hyphenation, such as: (1) syllable-based hyphenation, (2) compound words, (3) morphological determination, (4) context-sensitive hyphenation, (5) avoidance of stylistically undesirable hyphenations, (6) prevention of ambiguities or unintended meanings, (7) semantic-based hyphenation, and (8) universal syllabic hyphenation, among others.

Future research could explore hybrid approaches that combine rule-based and machine learning techniques to further optimize hyphenation patterns. Additionally, expanding the dataset to include more diverse Portuguese corpora could help address rare edge cases and improve the overall robustness of the hyphenation system. We believe the findings

of this study provide a solid foundation for ongoing improvements in TEX hyphenation rules, ultimately contributing to better automated text processing in the Portuguese language.

References

[1] Aulete. Aulete online dictionary. aulete.com.br/

[2] D.P. Cegalla. *Novíssima Gramática da Língua Portuguesa*. Companhia Editora Nacional, 2020.

[3] C. Cunha, L. Cintra. *Nova gramática do português contemporâneo*. Lexikon Editora Digital, 2016.

[4] A.M. de Araújo, T. Maruyama. A hifenização em português. *Idioma*, (28):90–107, 2015.

[5] P. de Magalhães Gândavo. *Regras que ensinam a maneira de escrever e orthographia da lingoa portuguesa, etc.* Lisboa, 1574. purl.pt/324

[6] P.J. de Rezende. Portuguese hyphenation table for TEX. *TUGboat* 8(2):102–102, 1987. tug.org/TUGboat/tb08-2/tb18software.pdf

[7] P.J. de Rezende, J.J.D. Almeida. Hyphenation patterns for Portuguese. mirror.ctan.org/language/hyph-utf8/tex/generic/hyph-utf8/patterns/tex/hyph-pt.tex.

[8] Dicio. Dicio online dictionary. www.dicio.com.br/

[9] Y. Haralambous. A revisited small tutorial on Patgen, 28 years after, 2021. ctan.org/pkg/patgen2-tutorial

[10] Hunspell team. Hunspell. hunspell.github.io/

[11] Hunspell team. Hunspell hyphenation library. github.com/hunspell/hyphen/

[12] D.E. Knuth. Preliminary preliminary description of TEX. TEXDR.AFT, May 13, 1977. www.saildart.org/TEXDR.AFT[1,DEK])]

[13] R. Levien. *Brief explanation of the hyphenation algorithm herein*. Hunspell, 1998. github.com/hunspell/hyphen/blob/master/README.hyphen

[14] F. Liang, P. Breitenlohner. PATtern GENeration program for the TEX82 hyphenator. Source code of Patgen. mirror.ctan.org/info/knuth-pdf/other/patgen.pdf

[15] F.M. Liang. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Stanford University, 1983. tug.org/docs/liang/

[16] Linguateca. Cetenfolha corpus. www.linguateca.pt/cetenfolha/

[17] Michaelis. Michaelis online dictionary. michaelis.uol.com.br/

[18] L. Németh. Automatic non-standard hyphenation in OpenOffice.org. *TUGboat* 27(1):32–37, 2006. tug.org/TUGboat/tb27-1/tb86nemeth.pdf

[19] Palavras NET word list. www.palavras.net/

[20] TEX pattern authors. TEX hyphenation patterns. tug.org/tex-hyphen/

[21] Portal da língua portuguesa online dictionary. www.portaldalinguaportuguesa.org/

[22] Priberam. Priberam online dictionary. dicionario.priberam.org/

[23] K. Scannell. Hyphenation patterns for minority languages. *TUGboat* 24(2):236–239, 2003. tug.org/TUGboat/tb24-2/tb77scannell.pdf

[24] Senado Federal. *Acordo ortográfico da língua portuguesa: atos internacionais e normas correlatas*. Apr. 2013.

[25] P. Sojka. Notes on compound word hyphenation in TEX. *TUGboat* 16(3):290–297, 1995. tug.org/TUGboat/tb16-3/tb48soj2.pdf

[26] P. Sojka. Notes on compound word hyphenation in TEX. Technical report, Masaryk University in Brno, Faculty of Informatics, August 1995.

[27] P. Sojka. *Competing Patterns in Language Engineering and Computer Typesetting*. Ph.D. thesis, Masaryk University, Brno, 2005. www.fi.muni.cz/usr/sojka/papers/sojka-thesis.pdf

[28] P. Sojka. Hyphenation — a tutorial for TEX users. www.fi.muni.cz/~sojka/PB029/hyptut.pdf, 2002

[29] P. Sojka, D. Antoš. Context sensitive pattern based segmentation: A Thai challenge. In *Proceedings of EACL 2003 Workshop on Computational Linguistics for South Asian Languages — Expanding Synergies with Europe*, pp. 65–72, Budapest, 2003. www.fi.muni.cz/usr/sojka/papers/thaieacl.pdf

[30] P. Sojka, O. Sojka. The unreasonable effectiveness of pattern generation. *TUGboat* 40(2):187–193, 2019. tug.org/TUGboat/tb40-2/tb125sojka-patgen.pdf

[31] Wikcionário. Wikcionário online dictionary. pt.wiktionary.org

[32] Wikipedia. Portuguese wikipedia dump. dumps.wikimedia.org/ptwiki/latest/

◇ Leonardo Araujo
MG 443, Km 7 Fazenda do Cadete
Ouro Branco, 36495-000
Brazil
leolca (at) ufsj dot edu dot br
https://sites.google.com/site/leolca/
ORCID 0000-0003-3884-2177

◇ Aline Benevides
benevides dot aline12 (at) gmail dot com
ORCID 0000-0003-1814-593X

Identifying glyphs in some 16th century fonts: Hochfeder's font no. 2, a case study

Janusz S. Bień

Abstract

Selected characters from 16th century fonts are discussed in the paper, notably Kaspar Hochfeder's font no. 2. In particular their representation in the Unicode standard is investigated. For some of them the encoding is obvious, but there are also characters for which their encoding remains an open question. Some old characters don't have Unicode codepoints, but can be printed with an appropriate OpenType/TrueType fonts using typographic features. Two such fonts are used: Junicode and Incunable. Some characters from Gutenberg's bible are also mentioned.

1 Introduction

As the starting point we use here Kasper Hochfeder's font number 2. The table of its glyphs, prepared by Maria Błońska and Anna Wolińska, was included in the plate 20bis of the fascicule *I* (the second edition) of *Polonia Typographica Saeculi Sedecimi* [10][1] (the total number of fascicules is 12). This fascicule is not available freely online, but the table (together with other similar tables) has been uploaded by me to the repository `github.com/jsbien/early_fonts_inventory`. The table is also included here as Fig. 1.

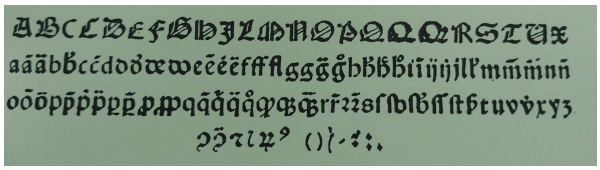


Figure 1: Hochfeder's font 2

In this paper we limit our attention to the letters and brevigraphs (abbreviations) presented in Fig. 2.

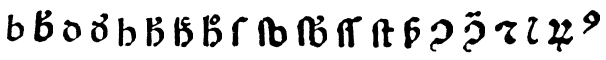


Figure 2: Hochfeder's font 2 — selected glyphs

The work [10] contains a list of publications with the information about the fonts used, and 34 of them reportedly use font no. 2; at this writing, nine of them have been digitized and are freely available.

It's not my goal to analyze the font usage, but for some problematic characters I tried to spot them in the texts.

[1] In [7] it was erroneously described as font 8 from plate 20.

2 The workflow

The character tables extracted by me from the scans or photos of *Polonia Typographica* plates were converted to DjVu. During the conversion the scan is split into three layers: the background, the binary mask and the foreground, while *the mask image is a high-resolution bilevel image (e.g., 300 dpi) and is typically where the text is stored* [13].

As described in [7], the djview4poliqarp indexes has been created. They were used to select the glyphs for the present paper.

A rather promising experiment segmented a font table into its individual glyphs, using code extracted from glyphtracer by Jussi Pakkanen (`github.com/jpakkane/glyphtracer`) and modified by ChatGPT 3.5 according to my specification. The Python script (`segment_character_table.py`, made available in the repository) handles correctly the diacritics over the letters, but is confused by the letter *M* (not surprising if you look at its shape), which is split into two glyphs. On the other hand some glyphs have not been separated; ChatGPT was unable to improve the splitting algorithm. These faults have to be corrected by hand or by some postprocessing, but this is not always necessary as for some purposes they are not relevant.

I am still a believer in using the DjVu shapes dictionary to spot interesting characters, as described in [7], but this time I have not used this approach, despite the fact that a new tool by Robert Mast is now available (`github.com/rmast/revealshapes`). The reason is the nature of the scans which would require substantial preprocessing (Fig. 3).

In consequence I used a simple technique, namely browsing the text with the already mentioned program djview4poliqarp, and storing the references to the interesting fragments in an index. Later the index was processed to extract the snippets from the scan and to generate LaTeX code to include them in the paper. The index has been uploaded to the repository mentioned earlier to allow interested readers to investigate the context of the snippets.

3 The Unicode standard

The current version of Unicode is 16.0.0, released in September 2024. It contains 154,998 characters.

To make a long story short, Unicode characters have only a rather loose relation to the characters we see in print or in writing (sometimes called user-perceived characters). In consequence I will refer to the item we want to encode as *textels* (I've used the term in informal texts for some time already) while the Unicode characters in the pure technical sense

doi.org/10.47397/tb/45-3/tb141bien-brevigraphshf

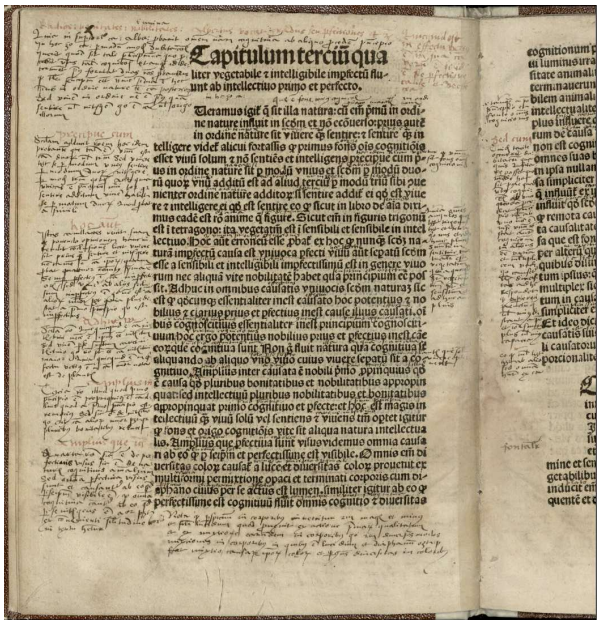

Capitulum tercium qua

Figure 3: A page from Albertus *De intellectu et intelligibili*, 1504

(in the standard they are also called *scalars*) I call *textons*.

There is also a useful notion of *typeme*, introduced by Jacques André and discussed by me in [6]. The difference between *typeme* and *textel* is rather subtle and in many contexts they may be used interchangeably.

I think it would be useful to distinguish several kinds of textels. The textels which can be coded in one or several ways in Unicode may be called *coded textels*.

Let us review the important aspects of textons (scalars):

- Codepoint, i.e., a unique number; it is usually written in the hexadecimal form and prefixed by U+.
- A unique name which is a more or less conventional label, e.g., OX, EIGHT PETALLED OUTLINED BLACK FLORETTE, ARABIC LIGATURE UIGHUR KIRGHIZ YEH WITH HAMZA ABOVE WITH ALEF MAKSURA ISOLATED FORM.[2].
- The semantics, i.e., a set of properties (the codepoint and the name are just a start).
- The representative glyph, i.e., the visual form of the texton; we will call it also the default glyph.

Later we will mainly see examples of textels encoded by single textons. We will see also an example of a textel encoded as so-called *extended graphemic cluster*, which I consider a misnomer (some regular Polish letters are *extended graphemic cluster*s according to the definition). This consists of a sequence of textons, namely the base character followed by some number of the so-called *combining characters*. The main properties of such a sequence are inherited from the base character.

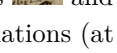
We will see also an example of using tag characters for encoding glyph variants. The tags have no glyphs, so are in principle invisible, but for documentation purposes they can be visualized as, e.g., 🏷 means the character U+E0031 TAG DIGIT ONE.[3]

We will see also some examples of the Private Use Area codepoints assigned to specific characters or glyphs by the Medieval Unicode Font Initiative (MUFI, `mufi.info`). For these codepoints we use the prefix M+.

Occasionally we will refer to a Private Use Area codepoint assigned a glyph only in the Junicode font. We will use for it the prefix J+ (if we don't want to specify the font we can use the prefix f+).

We will see also examples of using OpenType features to obtain glyphs which have no codepoint assigned in either the Unicode standard proper or in the Private Use Area. We will follow the feature notation used in [3], but with an extension: the character or the string to which the feature is applied will be quoted in braces immediately after the feature value, e.g., `cv38[2]{s}`, which means the value 2 of the character variant feature no 38 applied to the letter *s*. The glyphs represented this way may be called *font textels*.

4 An overview of the selected characters

1. The textel 𝖻 is evidently U+0062 LATIN SMALL LETTER B ('b'), included here only to contrast it with the textel discussed in the next section.
2. The textel �support. is used in such words as  and  , which are respectively abbreviations (at least in the given context) for *substantia* and *substantialis*.[4] However there is a single example where the textel is equivalent to *b* ([5, p. 8]); it may be a printer's mistake.

   The encoding was discussed in the context of the ligature presented below in section 11. In consequence the proposed encoding is U+0062

[2] The first (U+1F402) and the last (U+FBF9) example are respectively the shortest and the longest names (`gist.github.com/js-choi/320275d05d6f252f6bd55199f76489a6`)

[3] The Emacs editor since version 29.1 has a so-called minor mode `glyphless-display-mode`, which seems quite useful.

[4] The reading provided by Gionata Brusa and confirmed by Christoph M. Winterer and Stefano Recchia in the Facebook Paleography Society Group (`www.facebook.com/groups/7687162686/posts/10159558549692687/?comment_id=10159558561077687&notif_id=1716444009397852`).

LATIN SMALL LETTER B with U+0335 COMBINING SHORT STROKE OVERLAY (‘ƀ’).

3. The textel ꝺ seems to be M+A77A LATIN SMALL LETTER INSULAR D (the MUFI sample glyph is ‘ꝺ’, but the Junicode font also provides the glyph ‘ɒ’ accessed as `ss11{d}`); it is included here primarily to contrast it with the textel discussed in the next section.
4. The textel ð. is a well-known brevigraph which occurs often in many texts. Erin Blake [8] calls it *d with two ascenders* and gives the examples presented in Fig. 4.

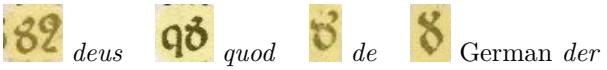
 *deus* *quod* *de* German *der*

Figure 4: Erin Blake’s examples of *d with two ascenders*

The encoding of this brevigraph is not obvious; it was discussed already in 2022.[5] The best option seemed to be treating it as M+F159 LATIN ABBREVIATION SIGN SMALL DE despite the fact that the MUFI sample glyph looks slightly different: ‘ð’. It is worth mentioning that there is a font which has a glyph with the shape quite close to the one in Hochfeder’s font; see Fig. 5. This is the Incunable font, ordered by Polish National Library for printing *Catalogue of Incunabula in the National Library of Poland*, and used in the first volume of the work, which is available online (`polona.pl/preview/e7be38b6-6d3a-4ba7-8579-15b14f73e1d9`).

‘ð’ (`ss01{ð}`) ‘ȝ’ (`ss02{ç}`) ‘ȝ́’ (`ss02{ḉ}`)

Figure 5: Some glyphs from the Incunable font

5. As for the textel h, it is U+0068 LATIN SMALL LETTER H (‘h’), although a contemporary reader can easily confuse it with *b*. It is included here only to contrast it with the textel discussed in the next section.
6. The textel ꞛ Erin Blake [8] calls *h with a tick on top* and gives the examples presented in Fig. 6.

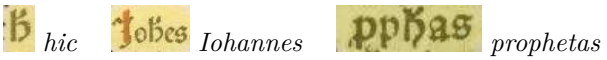
 *hic* *Iohannes* *prophetas*

Figure 6: Erin Blake’s examples of *h with a tick on top*

The encoding was discussed in 2022[6] and it was suggested to use U+0068 LATIN SMALL LETTER H followed by U+0335 COMBINING SHORT STROKE OVERLAY: ‘ħ’. In Junicode a glyph variant can be accessed with typographic feature `cv16[1]{ħ}`: ‘ꜧ’.

7. The textel ꞓ is in my opinion the letter *h* with a small letter *c* over its arch.[7] This interpretation was confirmed on the Facebook Paleography Society Group; in particular, Diane Warne Anderson interpreted the single letter word ꞓ (at least in the given context) as *haec*.[8]

   The glyph shape can be roughly approximated with U+0368 COMBINING LATIN SMALL LETTER C, which gives ‘hͨ’. The shape should be improved but it is not clear how to do it. A brute force approach is to use a special private use character, as done by MUFI for the character discussed in the next section.
8. The textel ꞕ is the letter *h* with a small letter *o* over its arch. This interpretation was also confirmed on the Facebook Paleography Society Group (cf. previous footnote); in particular, Christoph M. Winterer interpreted the single letter word ꞕ (at least in the given context) as *habeo*. The proposed encoding is M+F069 LATIN SMALL LETTER H WITH LATIN SMALL LETTER O ABOVE (‘ℏ̊’).
9. The textel ſ is U+017F LATIN SMALL LETTER LONG S (‘ſ’), included here only to contrast it with the ligatures discussed below. It is worth noting that in the Junicode font the glyph is available also as the letter *s* with the typographical feature `ss38[3]`.
10. The textel ﬞ is a ligature of long *s* and the letter *b*. There is no code point assigned to it, but the Junicode font creates this ligature automatically: ‘ſb’, although it is not documented explicitly.[9]
11. The textel ﬠ is the ligature of long *s* and the glyph described in section 2. The encoding was discussed in 2023.[10] The proposed encoding to be used with the Junicode font is the sequence U+017F LATIN SMALL LETTER LONG S, U+0062 LATIN SMALL LETTER B and U+0335 COMBINING SHORT STROKE OVERLAY: ‘ſb̵’.

[5] `github.com/psb1558/Junicode-font/discussions/133`
[6] `github.com/psb1558/Junicode-font/discussions/134`
[7] There seems to be no generally accepted terminology; I use the term from `typomil.com/anatomy`.
[8] `www.facebook.com/groups/7687162686/posts/10159538297237687/?comment_id=10159539385012687&reply_comment_id=10159539730067687`
[9] The best notation for documenting it is not clear. Perhaps something like `+{f}{b}`?
[10] `github.com/psb1558/Junicode-font/discussions/233`

12. The textel ײַ is a ligature of long *s* with long *s*. It is known as M+EBA6 LATIN SMALL LIGATURE LONG S LONG S (‘ײַ’). In Junicode, again the ligature is created automatically, but not documented explicitly.
13. The textel ﬞ is a the ligature of the long *s* with the letter *t*. It is known as M+FB05 LATIN SMALL LIGATURE LONG S T (‘ﬞ’). In Junicode, again the ligature is created automatically, but not documented explicitly.
14. The glyph ꝿ is in my opinion the brevigraph known as M+E8B7 LATIN SMALL LETTER LONG S WITH FLOURISH (‘ꝿ’).
15. The textel ↄ can be interpreted as U+2184 LATIN SMALL LETTER REVERSED C (‘ↄ’), which in the Unicode standard is unified with the Latin abbreviation for *con*, *com* and *-us*.

    A glyph better approximating the one in Hochfeder's font is provided by the Incunable font (Fig. 5).
16. The textel ↄ̈ can be interpreted as U+2184 LATIN SMALL LETTER REVERSED C (ↄ) with U+0308 COMBINING DIAERESIS (◌̈): ‘ↄ̈’.

    Mark Thakkar and Andrea Puglia (later also Mari Lessa and Er Orzo) on the Facebook Paleography Society Group[11] interpreted the brevigraph as *contra* (in the given context).

    A glyph better approximating the one in Hochfeder's font is provided by the Incunable font (Fig. 5).
17. The textel ꝛ is definitely U+204A TIRONIAN SIGN ET with the default glyph ‘⁊’.

    The Junicode font provides several alternative glyphs, more or less similar to the one in Hochfeder's font, namely: ‘⁊’ (`cv69[6]{⁊}`) and ‘ꝛ’ (`cv69[1]{⁊}` or J+F001D).
18. The textel ꝯ in my opinion is to be interpreted as U+A76D LATIN SMALL LETTER IS (‘ꝯ’).
19. The textel ꝓ is obviously U+A75D LATIN SMALL LETTER RUM ROTUNDA with the default glyph ‘ꝓ’.
20. The textel ꝰ is definitely U+A770 MODIFIER LETTER US with the default glyph ‘ꝰ’.

5 A problematic diacritical mark

There are six fonts containing all four glyphs we want to discuss here; see Figs. 7–9. In each of those fonts at least three of the glyphs have, in my opinion, an identical or at least very similar diacritical sign. The mark bears a very strong resemblance to a comma, so the question is whether the Unicode character U+0315 COMBINING COMMA ABOVE RIGHT can be used to represent it (U+0313 COMBINING COMMA ABOVE can be also considered, but it seems to me less suited for the purpose).

Figure 7: Hochfeder's fonts nos. 2, 3, 5 and 6

Figure 8: Haller's font no. 4

Figure 9: Ungler's font no. 1-1

Over 40 fonts listed in *Polonia Typographica* contain at least one of the four glyphs presented above; as the total number of the font tables is 75, this is not a marginal phenomenon. Figs. 10–13 show how the glyphs look in other fonts. The printer names (Augezdecki, Haller, Hochfeder, Ungler's first and second printing house) are abbreviated and continuous numbers are added for reference purposes.

[11] `www.facebook.com/groups/7687162686`

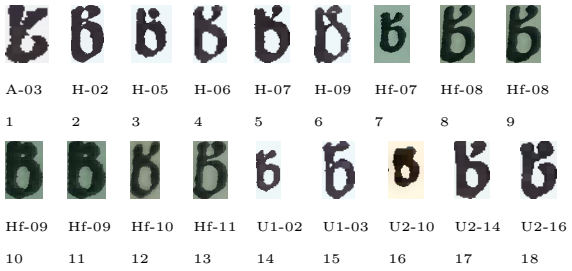


Figure 10: Modified *b*

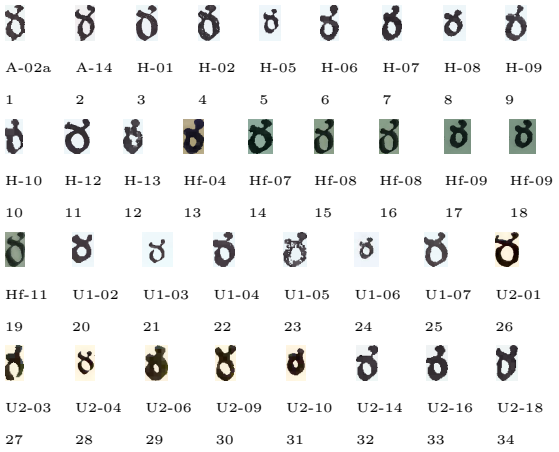


Figure 11: Modified *d*

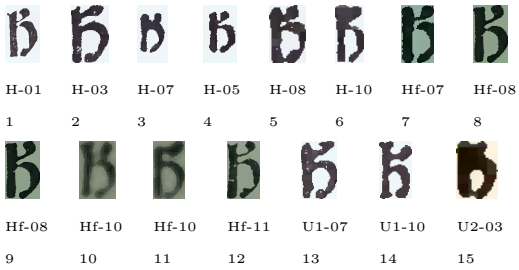


Figure 12: Modified *h*

Peter Baker in [3, p. 64] in section 6.1 (*A preliminary note on transcription*) writes:

> Converting a document from visual image to Unicode-encoded text adds a good bit of value all by itself, but only if done with due regard for the semantics of Unicode characters. Every Unicode character has a meaning, and that meaning is a help to readers. Using the wrong character is a hinderance to readers, even if it looks right.

I agree with this statement, so the question is, what are the semantics of the characters U+0335 COMBINING SHORT STROKE OVERLAY and U+0315 COMBINING COMMA ABOVE RIGHT? The paleographic literature known to me does not mention them. In Unicode they are present since version 1.1.0 of June 1993. There is no documentation describing the reasons for including them in the standard, so we have to look for other sources of information.

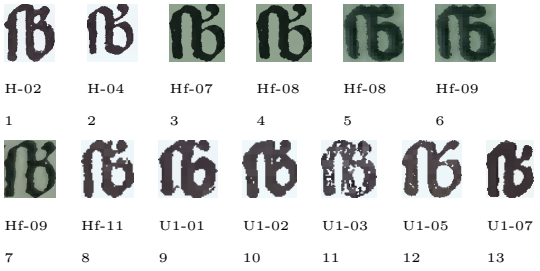


Figure 13: Modified *sb*

The AI chat *perplexity*[12] on June 3, 2024 said

> The Unicode character "COMBINING SHORT STROKE OVERLAY" (U+0335) is a combining diacritical mark used to represent a short horizontal line or stroke placed above a base character. Its primary purpose is for linguistic and phonetic notation systems, where it can indicate various modifications or annotations to the pronunciation or meaning of a letter or symbol. Some specific uses of the "COMBINING SHORT STROKE OVERLAY" include:
>
> In the International Phonetic Alphabet (IPA), it is used to indicate a mid-centralized vowel, such as in the transcription of certain English dialects.
>
> In the Uralic Phonetic Alphabet, it represents palatalization of a consonant.
>
> In mathematical notation, it can be used to indicate a vector or a specific operation or transformation applied to a symbol.
>
> In linguistic transcription systems, it may represent various phonetic or phonological features, such as nasalization, length, or stress patterns.

and

> The Unicode character "COMBINING COMMA ABOVE RIGHT" (U+0315) is a combining diacritical mark used to represent a comma-like symbol placed above and to the right of a base character. Its primary use is in linguistic transcription systems and phonetic alphabets to indicate specific sounds or modifications to the pronunciation of a letter. Some examples of its usage include:
>
> In the International Phonetic Alphabet (IPA), it is used to represent a rising-rising tone, such as in the Cantonese word "si" (詩, poetry).
>
> In the Americanist phonetic notation, it represents an extra high tone or a high-rising tone.
>
> In the Uralic Phonetic Alphabet, it indicates palatalization of a consonant.

[12] `www.perplexity.ai`

> In the Sinological tradition, it represents a rising tone in Chinese transcription systems.

As the reader may know, *perplexity* is supposed to provide sources of the information given, but in this case the sources listed were of little relevance. Thus verifying this information appears rather difficult, but I nevertheless think the information is generally correct, i.e., both characters are used in phonetic transcription.

So in my opinion, for our purposes using the stroke overlay has no advantage over the comma above.

Erin Blake in a private mail of June 27, 2024 pointed me to a passage in Capelli's abbreviation dictionary [9] which in English translation[13] has the form

> ... an oblique line, frequently hooked at one or both ends, cutting transversely across almost any letter of the alphabet, and assuming a variety of different meanings.

On the other hand, Peter Baker wrote on April 26, 2022:[14]

> The flourish on the right end of the horizontal stroke can be added to any horizontal stroke (in MSS and early prints).

I understand he means the stroke described in the above quotation of Capelli. He wrote also:

> If it's needed I could make it a variant of U+0335 [i.e., COMBINING SHORT STROKE OVERLAY].

Fortunately there is no deadline for making the decision and the question of how to encode the glyphs discussed in this section may be left open.

6 Some similar characters

In my earlier papers (e.g., [5]) I identified some of the characters discussed above with the characters used already in Gutenberg's bible. It does not seem now a good idea, but for completeness I will elaborate on this topic below.

It seems that the inventory of Gutenberg's characters was investigated first by Paul Schwenke [12]; see Fig. 14. His work was quickly superseded by the book by Gottfried Zedler [14]. The book was reprinted several times by Forgotten Books,[15] which offers paid access online with a free preview. In 2018 it was reprinted by Wentworth Press [11]. A copy of this edition was digitized by Google and made available by Internet Archive; unfortunately some

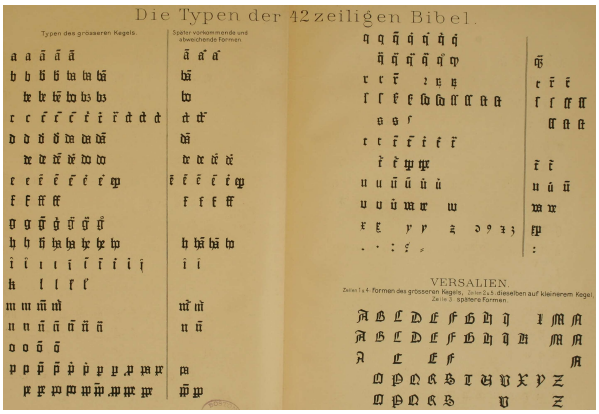


Figure 14: Schwenke [12, p. 32–33]

of the pages containing the character tables are distorted (perhaps a damaged copy or scanning fault). Zedler described also other fonts used by Gutenberg, in particular the one used to typeset so-called *Türkenkalender* (a pamphlet entitled *Eyn manung der cristenheit widder die durken*, which means "A warning for Christianity against the Turks"),[16] presented in Fig. 15.

Figure 15: Zedler [11],Tab. II b

Personally I found very useful the unpublished text [4] kindly provided to me by the author. (It is attached by the Printing Museum in Lyon to digital copies of a folio of the Gutenberg Bible purchased by the visitors.) He provided me also with some interesting information in private communications, for which I'm very grateful.

Jacques André in [1] discusses the relation of Gutenberg's characters to the Unicode standard and the Medieval Unicode Font Initiative. For some characters he also provides examples of their usage.

Although the modified letter *b* was used already in Gutenberg's 42-line Bible, it seems it still has no name and even no generally accepted description. In [1, p. 12], André proposes the name *latin small letter b with flourish* (he considers also an alternative one *latin small letter b ligated with arm of latin*

[13] hdl.handle.net/1808/1821
[14] github.com/psb1558/Junicode-font/discussions/134
[15] forgottenbooks.com
[16] bavarikon.de/object/bav:BSB-HSS-00000BSB00018195

small r by analogy to the modified letter *h* discussed later). It has neither a Unicode nor a MUFI codepoint assigned or even proposed.

According to Gerald Bettridge [4] it means *bis* and, after the long *s*, *ub*; see Fig. 16, also [1, p. 12] and [2, p. 12].

Figure 16: Modified letter *b* in Gutenberg's bible: *sublime* and *substantia* (Bodleian Library copy, page 21 and 64 of volume II)

The modified letter *h* is included in the MUFI specification as M+E8C3 LATIN SMALL LETTER B LIGATED WITH ARM OF LATIN SMALL LETTER R ('ℏ'), the upper case version is almost identical: M+E8C2 LATIN CAPITAL LETTER B LIGATED WITH ARM OF LATIN SMALL LETTER R ('ℏ'). In the Junicode font the glyphs are also accessible with the historical ligatures feature: hlig{hr}, hlig{Hr}.

Jacques André in [1, p. 17] gives an example of its use in the abbreviation of *philosophus*. Another example was provided by Bettridge; see Fig. 17.

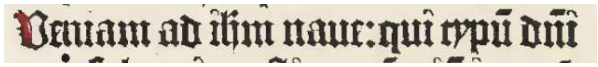


Figure 17: Veniam ad iesum [ihesum] nave . . . (Bettridge's example)

These characters does not seem to be used in Polish early prints, at least they are not listed in any of the fascicules of *Polonia Typographica*.

7 Final remarks

As the time of this writing (October 20, 2024) my proposal to use COMBINING COMMA ABOVE RIGHT instead of COMBINING SHORT STROKE OVERLAY for the textels in question, posted on June 16, 2024 on the Junicode site,[17] has no comments.

Some of the ideas discussed here were presented in the talk in Polish delivered on October 7, 2024 at the Natural Language Processing Seminar of the Institute of Computer Science — Polish Academy of Sciences (the Linguistic Engineering Group). The slides are available at `zil.ipipan.waw.pl/seminar` and `www.researchgate.net/publication/384676773`, and the video recording is available on YouTube on the seminar channel (`youtube.com/ipipan`).

[17] `github.com/psb1558/Junicode-font/discussions/284`

Acknowledgments. Thanks to Barbara Beeton and Karl Berry for their careful reading of the manuscript and improving the presentation and English wording.

References

[1] J. André. Inventaire des typèmes de la B42. Note de travail NT-1, Projet d'Inventaire des Caractères Anciens, 2015. `jacques-andre.fr/PICA/B42-typemes.pdf`

[2] J. André. Inventaire des typèmes latins et français existant dans Unicode/MUFI ou à y faire entrer. Note de travail NT-2, Projet d'Inventaire des Caractères Anciens, 2022. `jacques-andre.fr/PICA/SIGMA-PICA.pdf`

[3] P. Baker. Junicode — the font for medievalists. specimens and user manual for version 2.209, 2024. `github.com/psb1558/Junicode-font/`

[4] G. Bettridge. How to read the Gutenberg Bible. Text prepared for the Lyon Printing Museum.

[5] J.S. Bień. Repertuar znaków pisma nr 1 pierwszej drukarni Unglera (1510–1516) na podstawie Polonia Typographica. *Acta Poligraphica*, pp. 1–20, 2021. `www.cobrpp.com.pl/actapoligraphica/uploads/pdf/AP2021_Bien.pdf`

[6] J.S. Bień. Towards an inventory of old print characters: Ungler's *Rubricella*, a case study. *TUGboat* 44(3):364–375, 2023. `doi.org/10.47397/tb/44-3/tb138bien-rubricella`

[7] J.S. Bień. Basic Latin brevigraphs listed in *Polonia Typographica Saeculi Sedecimi* —Progress report. *TUGboat* 45(1):32–44, 2024. `doi.org/10.47397/tb/45-1/tb139bien-brevigraphspt`

[8] E. Blake. A briefing on brevigraphs, those strange shapes in early printed texts, 2021. `www.folger.edu/blogs/collation/brevigraphs/`

[9] A. Cappelli. *Lexicon abbreviaturarum. Dizionario di abbreviature latine ed italiane.* Ulrico Hoepli, 1929. `archive.org/details/CappelliDizionarioDiAbbreviature/`

[10] K. Piekarski. *Kasper Hochfeder, Kraków 1503-1505*, vol. 1 of *Polonia typographica saeculi sedecimi: Zbiór podobizn zasobu drukarskiego tłoczni polskich XVI stulecia.* Zakład Narodowy im. Ossolińskich, second ed., 1968. Maria Błońska, ed.

[11] E. Schröder, G. Zedler, J. Bauschinger. *Die älteste Gutenbergtype.* Wentworth Press, 2018. `archive.org/details/bub_gb_n8MB5Y6bz4AC`

[12] P. Schwenke. *Untersuchungen Zur Geschichte Des Ersten Buchdrucks.* Burg b. M.: Druck von A. Hopfer, 1900. `archive.org/details/untersuchungenzu00schw`

[13] Wikipedia. Djvu, 2024. `en.wikipedia.org/w/index.php?title=DjVu&oldid=1212632083`

[14] G. Zedler. *Die älteste Gutenbergtype.* Verl. der Gutenberg-Ges Mainz, 1902.

◇ Janusz S. Bień
Warsaw, Poland
`jsbien (at) uw.edu.pl`
`sites.google.com/view/jsbien`
ORCID 0000-0001-5006-8183

My website exploring language support and more in libre TrueType and OpenType fonts: https://typosetting.co.uk

Ken Moffat

Abstract

At typosetting.co.uk I have a website where I explore libre OpenType and TrueType fonts to try to see which languages they support (and to note mismapping of codepoints when I spot them). I am using XeLaTeX to produce PDFs showing the glyphs in the fonts, the languages I think they support, and some other comparisons between fonts.

1 Introduction

For many years I have been interested in using fonts which render all the text I am likely to see on mailing lists (whether or not I can read it), and whatever text is presented in the various rabbit-holes I dive into on Wikipedia.

My previous websites for doing this began in or before 2013. I had a C++ program to list what was in a font, and various Bash and Perl scripts. For the PDFs I was using LibreOffice, and at that time it indicated missing codepoints. In PDFs which compared files I referenced the font numbers.

Later, I added a second table (to avoid renumbering) for some other fonts, and then in 2023 I revisited fonts and added a third table.

Along the way, perhaps in 2016, LibreOffice had changed to plucking missing glyphs from other fonts when a font lacked a codepoint. Before that I had picked up a little information for testing LaTeX etc. to check that from-source builds worked, and determined that XeLaTeX mostly worked for me when displaying multiple languages.

For all languages I cover, I show Article 1 of the Universal Declaration of Human Rights to give a flavour of how text will look. For Latin and Cyrillic alphabets and monotonic Greek I attempt to show the whole alphabet including accented letters.

As a novice I used the default page layout with large margins and lots of space above and below the text. I ignored text that overflowed and did not think about any hyphenation.

My initial interest was only for on-screen output. At that time fontconfig was the tool that decided which glyphs to use. (Now browsers, office suites, and desktop environments can all override fontconfig.) As a result, I ignore any supporting files which might be on CTAN and can happily use a newer version of a font even when CTAN is out of date.

2 The new site: typosetting.co.uk

I prefer a readable dark-mode site without JavaScript. The PDFs have a white background as usual, and links to directories use a white background.

The three tables are now merged into one, keeping the same numbers. I have added indexes to link the fontconfig names to the table entries, and similarly to link my own identifiers (which omit spaces and are usually based on the filename). Figure 1 shows the first few rows of the table

That table is now very wide. I inserted a column showing when I revised the PDF for the supported languages, with links to the .tex file and a link for the small cap .tex file if there is one. The lines above the table, and on other pages, are much narrower, based on what looks good with my default sans font.

For fonts which do not cover Latin or Cyrillic alphabets or CJK languages I am not attempting to revise the pre-2024 documents. Although I have always used HarfBuzz with Graphite2, some of the old rendering of complex scripts might be poor [1].

I have not yet revised several CJK fonts where I am aware of newer versions. For any file revised before 2024 there is a possibility that it will no longer compile in XeLaTeX and there are probably typos in any Latin or Cyrillic alphabets, outdated Article 1 text in a few languages, and ugliness (formatting of CJK, and some of the TeX markup).

At the end of 2023 I realised that showing overlong lines was poor. For many languages with Latin alphabets polyglossia can usually do an adequate job of fitting the available space. But for Cyrillic alphabets, Greek, CJK and others the font needs a specific OpenType tag (cyrl, grek, hani, etc.). For Northern Sami (supposedly supported, but did not hyphenate) I found some guidance online for splitting the words into syllables. Similarly, for several Cyrillic languages where I was unable to use polyglossia I found various suggestions on how to split syllables. The results may be ugly to native readers.

For a few languages, and for some small caps fonts, I have to reformat the Article 1 text. Mostly I now use shorter lines to avoid putting vast spaces between words. For monospace fonts, much of the output includes spaces added automatically to justify the lines, but occasionally I will add non-breaking spaces to replicate what happens in a graphical terminal such as xfce4-terminal.

3 Characteristics I have noted

I have grouped fonts into sans, serif, and monospace, and then for Latin alphabets tried to separate the styles (transitional serif, modern serif, etc.). For monospace and sans I am no longer sure that such a

doi.org/10.47397/tb/45-3/tb141moffat-typosetting


font: Many fonts lack version numbers: for those I note the fontfile date or commit date.
Codepoints links to plain text files of what is in the font.
Coverage reports the codepoints within their Unicode blocks.

This table is very wide, you can scroll across but ideally use a wider window.
The identification numbers are repeated at the end of each line.

No.	Fontconfig	Identifier	Revised	Type	Filename	Glyphs	Languages	Bold	Italic	Package	License	Codepoints	Coverage	
1.001	Aerial Mono	AerialMono	2024/07/30 languages.tex	Monospace Sans	AerialMono.ttf	Glyphs	Langs	yes	yes	fonts-arkpandora_2.04 from Debian	Bitstream Vera	codepts	covrg.	1
1.002	Aerial	Aerial	2024/07/30 languages.tex	Neo-Grotesque Sans	Aerial.ttf	Glyphs	Langs	yes	yes	fonts-arkpandora_2.04 from Debian	Bitstream Vera	codepts	covrg.	1
1.003	Andika	Andika	2024/08/02 languages.tex sc.tex	Humanist Sans	Andika-Regular.ttf	Glyphs	Langs SC Langs	yes	yes	Andika from Google	SIL OFL v1.1	codepts	covrg.	1
										Arimo dated				

Figure 1: The start of the big table at `typosetting.co.uk`.

distinction is generally useful. In the past, particularly in my comparative "lipsum" files, I have made a similar distinction for types of Cyrillic fonts. For CJK fonts I am now tending towards merely labelling them as sans or serif — the terminology varies and is not always consistent for CJK fonts for a specific language.

Beyond that I show italics and bold weights for fonts offering Latin alphabets. For recent families I mention the range of weights, and briefly show Cyrillic italics.

For CJK fonts I hope to create language-specific files showing codepoints known to differ across languages, together with examples of each Noto CJK font and WenQuanYi Zen Hei (a preferred font in fontconfig). The first of these, `CJK-sans-sc`, is in the `files/PDF-cjk/` and `files/cjk-tex/` directories (`sc` for Simplified Chinese). I have few sans (Hei) sc fonts so this was an easier place to start, but many of the chosen codepoints look similar in sans fonts.

4 Validating my analysis, tools and templates

I doubt there is much of significance in any of my analysis of what a font covers, but in the last year I have read many articles online where research could not be validated, sometimes because the tools were not available. In addition, many fonts needed weird workarounds in some places, particularly for extended text in small caps. So I have attempted to make all my tools, and my templates, available.

4.1 Templates

For people using XeLaTeX and/or `polyglossia` (and perhaps for users of LuaLaTeX) the templates are perhaps the most interesting thing (apart from the `.tex` files for any fonts of interest to them). They currently include:

4.1.1 `languages-full.tex`

This is for attempting to see which languages a font covers. It includes a few languages which are covered by FreeSans and FreeSerif.

Beyond that, it has a lot of comments for me about things to look at when I revise an old file, and reminders about mentioning styles (normal, italic) and weights — in the past I tended to talk of font "faces" — followed by sections for the various languages.

If a language supports Vietnamese I attempt to show the added accents for the various tones. The individual letters are precomposed, but I show a separate combining accent at the start of the line to indicate what is used. A few fonts do not support those combining accents, in those cases I show their missing glyph indication. More commonly, the combining accent is either in the gutter (negative spacing needed) or ends up being inset (monospaced fonts). None of the PDFs have well-aligned columns for all of these accented letters.

For my revisions after the site went live I now try to be specific about how missing codepoints are rendered, but most files use older, more general, language like "blank spaces", "nothing", or "indication of a missing glyph".

For quotation marks I show all supported variants, but for currencies and some other symbols such as copyright I list them all whether or not they exist in that font. The output messages usually end with missing codepoints. I now know that I need to review those messages, particularly where HarfBuzz might add an accent or simulate a non-breaking hyphen.

The CJK area starts with a series of codepoints which may have different forms in some of the languages. That is to help me decide at which language a font is targeted, and will be deleted once I've made the decision. It then includes some English text to help me know where the margins should be (in theory the XeLaTeX CJK settings near the front of the file should fix that) which I will again delete.

In some cases, all I need to do for `fontspec` is use the name known to fontconfig; for XeLaTeX, unlike LibreOffice, this is the first name, before any comma, if there is more than one name. But for several fonts, such as Carlito, FreeSerif, lmroman and Junicode, I had to add a `Path` statement within

the setmainfont invocation to find one or more of the related font files.

4.1.2 languages-sc.tex for small caps

A very much cut-down template, for Latin and Cyrillic alphabets, and monotonic Greek. Many fonts with small caps omit certain less-common letters. For Azeri or Turkish (which use both dotted and dotless i) I was advised to use lowercase dotless i with combining dot above.

4.1.3 languages-italic.tex

A working document, to help me decide which languages have italics and/or small cap italics.

4.1.4 Hangul-Hanja.tex

Until recently I assumed that Hanja (South Korean Han) glyphs were no longer in common use. But it seems they are still used, both for place names on signs (probably only sans glyphs) and for both surnames and given names in official documents. I found a 2015 PDF of the list of permitted glyphs online, organized by Hangul syllable. But I cannot paste from that, and the only online pastable glyphs are mostly in Wikipedia, under Korean given names. Those pages are for notable people, so many of the permitted glyphs are not shown.

In addition, many of the glyphs are hard for me to distinguish unless I enlarge a text file until it is hard to navigate. As a result, these are only a subset of the permitted glyphs and probably contain some duplicates — certainly the initial files I created have a few duplicates.

The aim of this is to let me say "seems to adequately cover Hanja used in names", or "covers most", "covers many", ...

4.2 Programs and scripts

These programs and scripts assume I am the only user on the machine, I make no attempt to use secure names for temporary files and am happy to write working files to known names in /tmp. These items have evolved since 2011, I do not normally touch most of them.

In files/tools/to-compile are directories for the following two tools and also source tarballs.

4.2.1 get_codepoints

The first requirement is to identify which codepoints are provided in a font. I found a has_char script by user repolho on github (github.com/repolho/has_char) which is public domain. It links to fontconfig and attempts to bypass the (obsolete?) rules that fontconfig uses to determine if a font has all the codepoints needed for the current locale.

I have been using this since 2016 with various releases of fontconfig and GNU C++, most recently fontconfig-2.15.0 and gcc-14.1.0.

4.3 getfonts

That get_codepoints script reads a TTF (normally, a regular or medium file, I suppose other files could have fewer codepoints). But historically fonts for Chinese and Japanese have been supplied as TTC (TrueType Collection) files to save space and fit within a size limit. I used to use fontforge to extract an individual TTF, but it was always very difficult to use (tiny faint text, very hard to read, and prone to errors while trying to extract a single font to a TTF).

If I now need to repeat the process I found a repo called textile at github, designed for copying TTFs from a TTC on Mac OS X (github.com/DavidBarts/getfonts). The program is called getfonts; I had to fiddle about a bit to get it working. On the TTC I tested, it did not find the font name, so I had four numbered fonts. But looking at fc-list showed me the names of each so it is good enough.

I had hoped to fork the original repo, then work out what I had changed and add the changes as individual commits, but time is not on my side.

4.4 create-codepoint-files

This script is how I get the lists of codepoints in a file. At times in the past I have had problems with things not found by get_codepoints, so as a backup I install ttfconfig.pl which is shipped in the examples part of Perl module Font-TTF-Scripts (metacpan.org/dist/Font-TTF-Scripts).

I have had special code to deal with known TTC files, but it looks as if that is no longer used and only a variable remains.

Then I merge both files, converting the codepoints to 0xXXXX or 0xXXXXX format so that I can sort them, then write them as U+XXXX or U+XXXXX. After that I run the font-contains script, described next, to create the coverage file which lists the codepoint ranges of this font within their blocks.

4.5 font-contains

This sources the unicode-blocks script (described below) and then converts the codepoint to decimal for less-slow shell maths — on a big font such as the main (Latin, Cyrillic, Greek) Noto or Source fonts, or the Noto CJK fonts, it will take several minutes. Unicode values with five hex digits are written to a temporary file for a second pass (probably unnecessary). It then uses the data from unicode-blocks to find

which block it should be in and writes lines of the contiguous codepoints present within that block.

4.6 `generate-all-characters`

This Bash script attempts to write every character in a font to a text file. That text file can then be opened in LibreOffice Writer, changed to use the required font, and formatted with page headings, footings and repeating the block names after the page breaks. If the 32 characters for a line (or to the end of the block if sooner) have gaps, spaces are used as padding.

For combining characters they are applied to a space, which usually works fairly well for Latin and Cyrillic combining accents. For more complex scripts the results may be poor (again see [1]) With CJK fonts I typically notice that Han glyphs are double-width, but that Hangul (Korean) seems to be an intermediate width.

4.7 Determining coverage of small caps

Most fonts with small caps include them as variants of the normal lowercase letters. Sometimes I can read what is in the file and generate the codepoints with some degree of confidence. I also report things I'm not sure about, which led me down rabbit holes for old Polish and for the Ukrainian yi-yi combination.

4.7.1 `find-small-caps`

A Bash script using the `otfinfo` program from Eddie Kohler's `lcdf-typetools` [2].

4.7.2 `merge-sc-codepoints`

This Bash script then merges the list of small caps codepoints into an updated codepoint file, adding '`+sc`' wherever there is a small cap available.

4.8 `unicode-blocks`

This is the data used by `generate-all-characters`: a list of the decimal values for the end of each block (strictly, the first value after this), a value for `MAXBLOCK` (378 as of Unicode 15.1.0 : unassigned blocks may be split in the future), and the block names with a list of their Unicode ranges. The process to create this is discussed in the next section.

4.9 Updating Unicode

When a coverage report shows codepoints in an unassigned area, it implies that Unicode has been extended. To update the data I use the following scripts:

4.9.1 `Blocks.txt`

I use the current list of Unicode assigned blocks. It can be downloaded from `unicode.org/Public/UCD/latest/ucd/`.

4.9.2 `update-blocks.sh`

This script processes `Blocks.txt` to create the file `unicode-blocks`. Although it invokes `/bin/sh` it might be Bash-specific.

It creates a temporary file, `assignments.txt`, and uses the `parse` script to get blocks and to create `unassigned` blocks to fill the gaps. Then it concatenates the files to get the `unicode-blocks` file.

4.9.3 `parse`

This invokes `/usr/bin/perl` to read through the `assignments.txt` file. At the end it appends the number of blocks.

5 Acknowledgements

Bob Tennent for suggesting `otfinfo -g` to list the small caps.

Don Hosek for suggesting dotless i with combining dot above for Turkish small caps.

Both Bruno Voisin and Herbert Voss suggested alternative approaches to listing the small caps; these are in the `tex-live` list archives for October 2023 (`tug.org/pipermail/tex-live/2023-October/`).

David Carlisle for explaining how HarfBuzz will add combining accents to base characters or alternatively use precomposed characters when combining accents are added.

Max Chernoff pointed to an alternative approach to typesetting a grid of characters showing all codepoints using LuaTeX; in the `tex-live` list archives for May 2024.

Werner Lemberg prompted me to dive deeper into CJK formatting. After discovering how to fix XeLaTeX for this, I later found draft W3C recommendations.

To all these people, and anyone I've missed — thanks for your help.

References

[1] Behdad Esfahbod. State of Text Rendering 2024. `behdad.org/text2024/`

[2] Eddie Kohler. LCDF Typetools. `www.lcdf.org/type/`

◇ Ken Moffat
`https://typosetting.co.uk`

(Ab)use of ligatures and other font features

Hans Hagen

Introduction

In early May 2024 Karl Berry forwarded me a mail by Nelson Beebe referring to a discussion on a font forum where surprising ligature replacement in OpenType fonts was mentioned in the perspective of abuse: fonts as programs. Anyone familiar with this technology will surely confirm that ligatures are a feature that can be abused, but also that in practice it seldom happens. So let's explore this a bit. Before I show some examples I'll summarize how TEX deals with ligatures. I'll also make clear that it is unlikely that ligatures are used for unexpected replacements.

Ligatures

When TEX adds characters to the horizontal list it can replace successive character pairs into a single character, a so-called ligature. We end up with a mix of character and ligature nodes. When the paragraph is broken into lines hyphenation is applied when it makes sense which in turn means that these ligatures (and kerns) are temporarily discarded for the word being checked. For that purpose the node that stores a ligature keeps track of the characters that made the ligature. Although it is not that relevant for what we discuss here, it is good to keep in mind that whatever sequence of characters we convert to another sequence of characters, hyphenation can be an influence and whatever we intend might not work well at a line break.

In LuaTEX we don't have character and ligature nodes but only glyph nodes that can be either a normal character or a ligature. We (can) still provide the components that made the ligature but in LuaMetaTEX we don't have these. If needed, the subtype can be used to distinguish between a character and ligature. We don't need the components because hyphenation has already happened before ligatures and kerns are applied.[1]

Although there are these differences between engines in how and when they construct (and in traditional TEX deconstruct) ligatures, in both cases they are the same: a multiple to singular mapping. The TEX engine looks at two characters. For example, an 'f' and 'i' become an 'fi' and an 'f' followed by a 'f' becomes 'ff'. Of course that only happens when a font has such a ligature. The new character 'ff' itself can combine with a following 'i' into a 'ffi'. So, even for a three character ligature we go via dual replacement, so for more complex ligatures we need intermediate ligatures or bogus characters.

Deep down TEX has several kinds of ligatures (the following description is taken from the LuaTEX manual). When TEX inserts a new ligature, it puts the new glyph in the middle of the left and right glyphs. The original left and right glyphs can optionally be retained, and when at least one of them is kept, it is also possible to move the new 'insertion point' forward one or two places. The glyph that ends up to the right of the insertion point will become the next 'left'. In this table the `|` indicates the final insertion point.

case	characters	ligature	outcome
0	`a|b`	`c`	`|c`
1	`a|b`	`c`	`|cb`
2	`a|b`	`c`	`|ac`
3	`a|b`	`c`	`|acb`
4	`a|b`	`c`	`c|b`
5	`a|b`	`c`	`a|c`
6	`a|b`	`c`	`a|bc`
7	`a|b`	`c`	`ab|c`

In the first four cases the current position (pointer in the node list) moves to the front, and in the last four to somewhere in the middle. This means that a ligature can combine with the next character into a new ligature. In practice only the first variant is used and all of these work on pairs. There are very few fonts in the TEX distributions that use the others. In general, the number of ligatures in traditional TEX fonts is small, simply because the 256 slots we have there are easily exhausted. But it shows that in principle complex replacements are possible.

In OpenType fonts we also have ligatures but here they are part of several substitution features.

substitute replace one character by one other character

alternate replace one character by one taken from a set of alternatives

ligature replace multiple characters by a single character

multiple replace a single character by multiple characters

In addition to simple replacements like these there are also contextual lookups that look at (a range of) characters, and are optionally triggered by what comes before and what comes after and then perform one of the above substitutions. It's a bit off-topic but TEX was written with Latin scripts in mind, so imagine what is needed to handle for instance Arabic with its much more complex ligatures. Fortunately hyphenation is not commonly used there

[1] An exception is Arabic where marks are anchored to shape ligatures.

doi.org/10.47397/tb/45-3/tb141hagen-ligatures

so we don't need to deal with ligatures crossing the pre-, post- and replacement parts of a discretionary.

In the table above we have a ligature substitution feature but that name is misleading. For instance an 'ff' ligature can indeed be a single glyph but quite often in OpenType it is implemented as two glyphs with some kerning, or with the second glyph being replaced by one that can blend with the first, which means that we end up with two glyphs with some negative kerning in between. The construction of ligatures can even be script- nd language-dependent. But still this combination of single replacement and positioning is called "ligature" and put under the `liga` feature. In TEX on the other hand ligatures are just multiple to single replacements.

As mentioned, the reason for this article is that there are users who fear that this ligature mechanism can be abused for malicious purposes. For instance, that we can replace one word by another word. A natural question for TEXies is then: does OpenType make users more vulnerable than when they'd use traditional fonts? As we explained before, it will be clear that even with simple fonts one can do unexpected replacements but in both cases a multiple to single replacement is not what achieves this: *we need a multiple to multiple replacement.*

How it can be done in OpenType

Let's start with OpenType. Here we can do a multiple to multiple replacement using a chain substitution. We use the ConTEXt (font loader) Lua interface to define a new feature.

```
fonts.handlers.otf.addfeature {
  name    = "mytesta",
  type    = "chainsubstitution",
  lookups = {
    {
      type = "multiple",
      data = {
        ["g"] = { "b", "a", "d" },
        ["b"] = { "g", "o", "o", "d" }
      },
    },
  },
  data = {
    rules = {
      {
        current = { { "g" }, { "o" }, { "o" },
                    { "d" } },
        lookups = { 1 },
      },
      {
        current = { { "b" }, { "a" }, { "d" } },
        lookups = { 1 },
      },
    },
  },
}
```

Here `current` is a list of characters to match and every position can have multiple matches. When there is a match the lookup sequence points back to the lookups table where one of the mentioned substitutions can take place. This newly created feature can now be applied:

```
\definefontfeature[mytesta][default]
  [mytesta=yes]
\definedfont[SerifBold*mytesta sa 1.2]
good or bad
```

Here the default feature sets enable ligature building (`liga`), kerning (`kern`) and other features considered standard. We also default to so-called node mode because instead of using the TEX engine, we delegate feature handling to Lua. The output is:

bad or good

Indeed the words `good` and `bad` get swapped. We can add a `before` and `after` sequence too and you can imagine that when this logic is translated into a binary OpenType format a regular user will not know that this can happen. So, indeed a font designer or someone who can intercept and replace a font can hide replacements like this. If we replace `mytesta` by `liga` it becomes part of the regular everyday ligature feature and because that one is often enabled by default it could go unnoticed for a long time. Because OpenType fonts can be huge and have lots of rules like these this cheat can be hidden very well. To some extent TEX users are somewhat better protected because they can always turn features on and off, while in less configurable situations defaults are used.

So, to summarize, in OpenType we do this: as we run over the list, at every character we check for the sequences `good` and `bad`. When we have a match we replace the matched characters by something else. The next variant shows a bit more complex setup with more lookups and different substitutions.

```
fonts.handlers.otf.addfeature {
  name    = "mytex",
  type    = "chainsubstitution",
  lookups = {
    {
      type = "substitution",
      data = {
        ["T"] = "t",
        ["E"] = "e",
        ["X"] = "x",
      },
    },
    {
      type = "multiple",
      data = {
        ["L"] = { "c", "o", "n", "t", "e", "x", "t" },
        ["l"] = { "c", "o", "n", "t", "e", "x", "t" },
      },
    },
```

```
  },
  data = {
    rules = {
      {
        current = {
          { "t", "T" }, { "e", "E" }, { "x", "X" }
        },
        lookups = { 1, 1, 1 },
      },
      {
        current = {
          { "l", "L" }, { "a", "A" },
          { "t", "T" }, { "e", "E" }, { "x", "X" }
        },
        lookups = { 2 },
      },
    },
  },
}
```

This might look a bit intimidating but it's just a way to enhance (or fix) a font and is in fact used in practice. This newly defined feature can now be applied:

```
\definefontfeature[mytex][default][mytex=yes]
\definedfont[SerifBold*mytex sa 1.2]
Using TeX (or TEX) logos, like LaTeX or (LaTEX),
can look a bit silly.
```

The output:

Using tex (or tex) logos, like context or (context), can look a bit silly.

Because we have control over the engine we can add more substitution features, not present in OpenType, because after all, control is what users love TEX for.

How it can be done in traditional TEX

In traditional TEX we don't have such a repertoire of replacements available but we can use a different trick. Again we use Lua to emulate it.

```
local function initialize(tfmdata,value)
  if value then
    local characters = tfmdata.characters
    local y = utf.byte("y")
    local e = utf.byte("e")
    local s = utf.byte("s")
    local n = utf.byte("n")
    local o = utf.byte("o")
    local cy = characters[y]
    local ce = characters[e]
    local cs = characters[s]
    local cn = characters[n]
    local co = characters[o]
    characters[y].ligatures = {
      [e] = { char = 255 }
    }
    characters[255] = {
      ligatures = {
        [s] = { char = 254 }
      },
    }
    characters[n].ligatures = {
      [o] = { char = 253 }
    }
    characters[254] = {
      width    = cn.width + co.width,
      height   = math.max(cn.height, co.height),
      depth    = math.max(cn.depth, co.depth),
      commands = {
        { "char", n },
        { "char", o },
      },
    }
    characters[253] = {
      width    = cy.width + ce.width + cs.width,
      height   = math.max(cy.height, ce.height,
                          cs.height),
      depth    = math.max(cy.depth, ce.depth,
                          cs.depth),
      commands = {
        { "char", y },
        { "char", e },
        { "char", s },
      },
    }
    tfmdata.properties.hasligatures = true
    tfmdata.properties.virtualized = true
  end
end

local specification = {
  name        = "myhacka",
  description = "myhacka",
  manipulators = { base = initialize, }
}
fonts.handlers.otf.features.register(specification)
```

Because in the end we need multiple characters we create a virtual font where a single slot (character) renders multiple characters. Of course one should also add proper kerning between the characters that make `no` and `yes`; copy the rightmost characters to the `kerns` table, and patch the ones that kern with the left character but we leave that as an exercise.

```
\definefontfeature[myhacka]
  [mode=base,myhacka=yes,liga=yes,kern=yes]
\definedfont[SerifBold*myhacka sa 1.2]
yes or no
```

Where in the OpenType example we delegated processing to Lua, here we let the TEX engine do the job. This is called base mode. Node mode picks up the data it needs for ligaturing and kerning from the font via Lua; in base mode it is part of the font definition that gets passed to the engine (like the `ligatures` tables here).

no or yes

In the previous example we do this at runtime but if you use for instance pdfTEX you need to make a real virtual font file (a `vf` file alongside the `tfm` file). Here the `commands` entry provides the recipe

for the virtual character where we draw from the same font.

So, to summarize this approach: we check each character for being the start of a ligature. When we have a match we replace the pair by a new character, in the case of `no` this is the final one, and in the case of `yes` an intermediate one that gets a follow-up check. Do you see the problem? If we enter this:

```
\definefontfeature[myhacka]
  [mode=base,myhacka=yes,liga=yes,kern=yes]
\definedfont[SerifBold*myhacka sa 1.2]
yes or no or yep
```

no or yes or ap

We get an unexpected replacement, because character 255 is already taken. The solution is to add this:

```
local function initialize(tfmdata,value)
  if value then
    local characters = tfmdata.characters
    local y = utf.byte("y")
    local e = utf.byte("e")
    local s = utf.byte("s")
    local n = utf.byte("n")
    local o = utf.byte("o")
    local cy = characters[y]
    local ce = characters[e]
    local cs = characters[s]
    local cn = characters[n]
    local co = characters[o]
    characters[y].ligatures = {
      [e] = { char = 255 }
    }
    characters[255] = {
      ligatures = {
        [s] = { char = 254 }
      },
      width    = cy.width + ce.width,
      height   = math.max(cy.height, ce.height),
      depth    = math.max(cy.depth, ce.depth),
      commands = {
        { "char", y },
        { "char", e },
      },
    }
    characters[n].ligatures = {
      [o] = { char = 253 }
    }
    characters[254] = {
      width    = cn.width + co.width,
      height   = math.max(cn.height, co.height),
      depth    = math.max(cn.depth, co.depth),
      commands = {
        { "char", n },
        { "char", o },
      },
    }
    characters[253] = {
      width    = cy.width + ce.width + cs.width,
      height   = math.max(cy.height, ce.height,
                          cs.height),
      depth    = math.max(cy.depth, ce.depth,
                          cs.depth),
      commands = {
        { "char", y },
        { "char", e },
        { "char", s },
      },
    }
    tfmdata.properties.hasligatures = true
    tfmdata.properties.virtualized = true
  end
end

local specification = {
  name        = "myhackb",
  description = "myhackb",
  manipulators = {
    base = initialize,
  }
}
fonts.handlers.otf.features.register(specification)

characters[255] = {
  ligatures = {
    [s] = { char = 254 }
  },
  width    = cy.width + ce.width,
  height   = math.max(cy.height, ce.height),
  depth    = math.max(cy.depth, ce.depth),
  commands = {
    { "char", y },
    { "char", e },
  },
}

\definefontfeature[myhackb]
  [mode=base,myhackb=yes,liga=yes,kern=yes]
\definedfont[SerifBold*myhackb sa 1.2]
yes or no or yep
```

no or yes or yep

It's trickier than this

It's clear that where in OpenType one can use different means than ligatures, in traditional TEX where we only have ligatures as a tool we have to combine them with another (powerful) magic: virtual fonts. In the end there is not much to worry about: access to the font and the user's machine is needed in order to achieve this. In fact, the previous examples are not even right.

There is one aspect that we didn't mention yet that makes an OpenType font more suitable. If we only want to replace words we need to add a boundary check so let's give an example of that.

```
local punctuation = { { " ", ".", ",", ":", ";" } }

fonts.handlers.otf.addfeature {
  name    = "myteste",
  type    = "chainsubstitution",
  lookups = {
    {
      type = "multiple",
      data = {
```

```
        ["g"] = { "b", "a", "d" },
        ["b"] = { "g", "o", "o", "d" }
      },
    },
  },
  data = {
    rules = {
      {
        before  = punctuation,
        current = { { "g" }, { "o" }, { "o" },
                    { "d" } },
        after   = punctuation,
        lookups = { 1 },
      },
      {
        before  = punctuation,
        current = { { "b" }, { "a" }, { "d" } },
        after   = punctuation,
        lookups = { 1 },
      },
    },
  },
}
```

This can now be applied:

```
\definefontfeature[myteste][default][myteste=yes]
\definedfont[SerifBold*myteste sa 1.2]
goody or badass and good or bad
```

goody or badass and bad or good

Getting this to work in TEX is hard because we run out of slots and because TEX in base mode (traditional) has no space. And even in OpenType we probably need to do more to get it right all times: punctuation, digits, all kinds of quotes, etc. So with the given TEX example we get even more unexpected changes because every work that contains the lookup replacement happens.

Unexpected ligatures in practice

Of course in practice fonts don't get hacked this way. But you can have surprises in some fonts. An example is Font Awesome Brands:

```
\definedfont[FontAwesome5Brands-Regular-400*default]
dropbox windows android apple linux wikipedia
```

Sometimes fonts have such goodies built in and as long as stylistic features are used for this it is just a handy feature (or you might call it a gimmick). Emoji fonts are good candidates for combining emoji into new ones, but changes in shapes (this could be driven by social or political influences) can also make for surprises. It demonstrates that it is always good to proofread what you typeset. The problem with such features is that when you decide to use another font, it is unlikely to work there so maybe it's best not to rely on it and use similar features that the macro package (might) provide instead.

A recent trend is to let monospaced fonts perform ligaturing, so we get for instance a 'less equal' symbol instead of a 'less' plus an 'equal'. And it can be highly annoying if that can't be turned off. Feature creep is probably more troublesome than an occasional nasty font, one that probably isn't worth using anyway.

Some final words

The suggestion in the post that fonts are programs is not entirely true because we're only dealing with lookups: it's data that we're talking about. That is not to say that fonts can't become programs. As explained in Niklaus Wirth's *Algorithms + Data Structures = Programs*, one needs the algorithms too and that's what the OpenType composition code does, with specific features either supported or not by a user interface or hard-coded assumptions. And although the descriptions of OpenType have improved over time, occasionally one has to ponder how to deal with complex combinations. And as usual bugs can become features. In that sense, the limitations of the TEX model are a protection against unexpected (but intended) side effects. If you nevertheless consider an OpenType font to be a program, you also should expect it to come with a manual explaining the reasoning behind the way it handles all this as well as showing the user what to expect, but I doubt that this is on the mind of a font developer.

So, while it is for instance no big deal in LuaTEX to add functions to the repertoire of features we can go further. At BachoTEX 2024 I was told that there are (premature) ideas about adding Web Assembly to fonts, which for sure will give some users the creeps, but then we're talking application-specific trickery which is outside our scope. The most recent color extensions are an example of undesired complexity and the question is if we want more of that. Maybe it makes sense on the web but on paper (and with TEX) we can often achieve the same independent of fonts that then need to have these programs embedded in every instance (one can wonder how updating and bug fixes works out then). Once we lock our workflows into libraries that themselves have dependencies we also enter the realm of programs with potential back doors, leaks, insecurity, compatibility issues, etc. But that's for another article. I only mention this in order to stress that the currently provided OpenType features are the least of our worries.

◇ Hans Hagen
Pragma ADE

Correcting for italic corrections

Hans Hagen, Mikael P. Sundqvist

Italic correction is missing from OpenType as a concept, so this is something that we had to deal with ourselves right from the start of the LuaTEX project.

This meant that after adding the `\glyphslant` primitive to LuaMetaTEX the automatic italic correction feature that ConTEXt provides had to be updated. Because we (Mikael and Hans) have been dealing with corrections for a while now in the perspective of math, we were discussing the updated heuristics that now have to deal with some more scenarios. As a test we used the Latin Modern italic and oblique fonts as well as the runtime slanted regular.

We were curious about the differences between the bounding boxes of the `f` in these alternatives, and noticed some interesting clashes between some upright shapes. And as test we then added the explicit italic correction `\/` directive and to our surprise the `fh` (or likewise `fB`) clashes went away. That made us look at the original and we noticed that there is italic correction in `cmr10` and its upright relatives. Here are examples without (top) and with (bottom) italic corrections applied:

The offhanded wolfhound ate many selfheals.
The offhanded wolfhound ate many selfheals.

This example is generated using pdfTEX and fonts with TFM metrics. So the question is: did you ever consider applying italic correction to these upright fonts? Of course something like `fh` is plausible because some fonts even have ligatures, but a correction between `gh` is another matter, right? Did you ever consider the possibility to force an italic kern in the middle of an upright sequence? And how about potential hyphenation? Or are they just one of these easter eggs?

In the transition to OpenType there could have been alternates that have the correction added to the width. Unfortunately we have now lost that correction information in those fonts, so we cannot tweak them at runtime.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University

Diagrams with TikZ: A practical example

Uwe Ziegenhagen

Abstract

I've been working on creating electronic music for some time and wanted to create a block diagram showing the signal path in a synthesizer. I used TikZ for this. The path to the finished diagram can perhaps also serve as an inspiration for some people to create more complex drawings with TikZ.

1 Basics and templates

Classic synthesizers usually consist of a series of interconnected VCOs (Voltage-Controlled Oscillators), VCFs (Voltage-Controlled Filters) and VCAs (Voltage-Controlled Amplifiers), which are controlled by so-called ADSR (Attack-Decay-Sustain-Release) envelopes. If you visualize their interaction in a block diagram, the image may be like the one in Figure 1, which was taken from Kent Lundberg's (MIT) website. This illustration will be the template for our TikZ journey today.

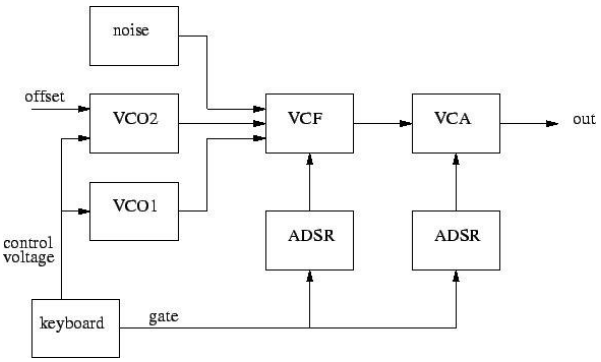


Figure 1: Our diagram template. Source: `web.mit.edu/klund/www/weblatex/node2.html`

2 Translation into TikZ code

In order to get a higher resolution version of the graphic and thus also have the opportunity to e. g. graphically emphasize individual parts, I decided to recreate it using TikZ. I wanted the implementation to be as close to the original as made sense.

My first step when using TikZ is always to draw a suitable grid that serves as the basis for the subsequent positioning work. TikZ has a `grid` option of the `\draw` command; the color, step size and line thickness of the grid can be easily adjusted.

I also put a "node" in the drawing. If you don't understand the term, you will find the appropriate explanation in the documentation at `tikz.dev/`

doi.org/10.47397/tb/45-3/tb141hagen-ic
doi.org/10.47397/tb/45-3/tb141ziegenhagen-diagrams1

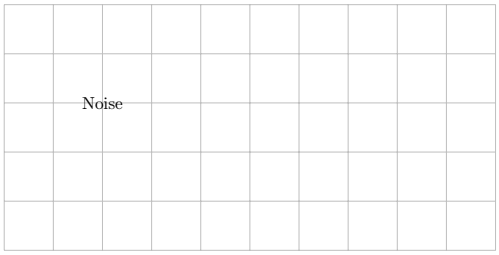


Figure 2: An individual node on a grid (scaled)

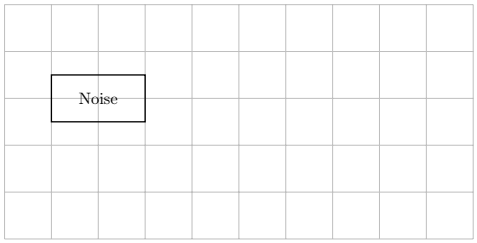


Figure 3: Grid and Node, style added

tikz-shapes, which I reproduce here in free translation: nodes are, in their simplest form, character strings that are set to coordinates. You can specify the coordinates — as I do in this example — but you can also calculate them based on e. g. proximity to other nodes. You can also give the nodes names, which makes later drawing operations much easier. They can have foreground and background colors as well as a shape. In short, they are wonderful little constructs that make drawing life much easier for us.

```
\begin{tikzpicture}
\draw[step=1cm,lightgray,thin](0,0) grid (10,5);
\node at (2,3) (noise) {Noise};
\end{tikzpicture}
```

Listing 1: Code for Figure 2

In the next step we take care of the external shape of the individual nodes. To do this, we define a style template in the options of tikzpicture, which we can then easily assign to individual nodes. I chose a black, rectangular shape, the minimum width of each node is set to 20 millimeters, the minimum height to 10 millimeters. See listing 2.

```
\begin{tikzpicture}[
box/.style={rectangle,thick,draw=black,
minimum width=20mm, minimum height=10mm}]
\draw[step=1cm,lightgray,thin](0,0) grid (10,5);
\node at (2,3) [box] (noise) {Noise};
\end{tikzpicture}
```

Listing 2: Code for Figure 3

This gives us everything we need to create and position the remaining nodes. It does not matter if individual nodes protrude beyond the grid, the grid

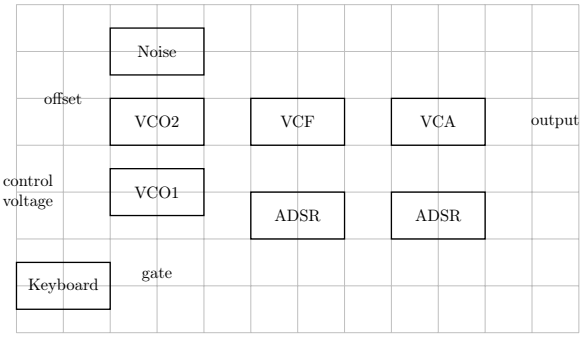


Figure 4: Figure showing the final nodes

will not be included in the final version of the drawing. For the two-line text we'll see in the "control voltage" node, I used a trick from tex.sx,[1] which uses a parbox whose width is set exactly to the length of the "control" word using the \widthof command from the calc package.

```
\begin{tikzpicture}[
box/.style={rectangle,thick,draw=black,
minimum width=20mm,
minimum height=10mm,align=center}]
\draw[step=1cm,lightgray,thin](-1,0) grid(11,7);
\node at (2,6) [box] (noise) {Noise};
\node at (2,4.5) [box] (vco2) {VCO2};
\node at (2,3) [box] (vco1) {VCO1};
\node at (5,4.5) [box] (vcf) {VCF};
\node at (8,4.5) [box] (vca) {VCA};
\node at (5,2.5) [box] (adsr1) {ADSR};
\node at (8,2.5) [box] (adsr2) {ADSR};
\node at (0,1) [box] (keyboard) {Keyboard};

\node at (0,5) (offset) {offset};
\node at (10.5,4.5) (output) {output};
\node at (2,1.25) (gate) {gate};

\node at (-0.75,3) (cv)%
 {\parbox{\widthof{control}}{
   control \\ voltage }};
\end{tikzpicture}
```

Listing 3: Code for Figure 4

Before we connect the individual nodes with arrows in the next step, let's take a look at the options that TikZ offers us. Listing 4 shows some of the possibilities. In addition to the direct lines (with -, a to b), there are also angular connections with |- for "first vertical, then horizontal" (c to d) or -| for "first horizontal, then vertical". We can also specify the angles at which the arrows exit a node or go into a node (g and h); even Bezier curves can also be used (i and j, the two control points are marked with a *). The last example (k and l) uses some of the specified

[1] tex.stackexchange.com/questions/31096/how-can-i-use-linebreak-inside-a-node-in-tikz

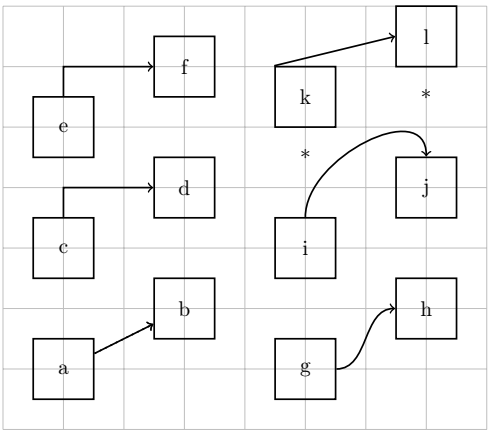


Figure 5: Digression: some possible node connectors

anchors in the individual nodes, which are addressed via the cardinal directions. See also the graphic in the instructions at `tikz.dev/library-shapes`.

```
\begin{tikzpicture}[
box/.style={rectangle,thick,draw=black,
minimum width=10mm,
minimum height=10mm,align=center}]
\draw[step=1cm,lightgray,thin] (0,0) grid (8,7);
\node at (1,1) [box] (a) {a};
\node at (3,2) [box] (b) {b};
\draw [thick,->] (a) -- (b);
\node at (1,3) [box] (c) {c};
\node at (3,4) [box] (d) {d};
\draw[thick,->] (c) |- (d);
\node at (1,5) [box] (e) {e};
\node at (3,6) [box] (f) {f};
\draw[thick,->] (e) |- (f);
\node at (5,1) [box] (g) {g};
\node at (7,2) [box] (h) {h};
\draw [thick,->] (g) to [out=0,in=180] (h);
\node at (5,3) [box] (i) {i};
\node at (7,4) [box] (j) {j};
\node at (5,4.5) (cp1) {*};
\node at (7,5.5) (cp2) {*};
\draw [thick,->] (i) .. controls (cp1) %
      and (cp2) .. (j);
\node at (5,5.5) [box] (k) {k};
\node at (7,6.5) [box] (l) {l};
\draw [thick,->] (k.north west) -- (l.west);
\end{tikzpicture}
```

Listing 4: Code for Figure 5

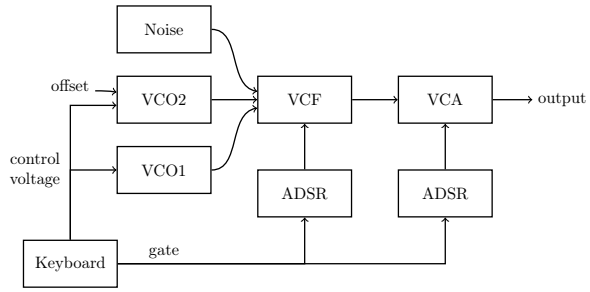


Figure 6: Final diagram

To connect the individual nodes together, we give additional `\draw` commands that receive the names of our nodes as parameters.

```
\draw[thick,->](vco1) to [out=0,in=190] (vcf);
\draw[thick,->](vco2) -- (vcf);
\draw[thick,->](adsr1) -- (vcf);
\draw[thick,->](adsr2) -- (vca);
\draw[thick,->](keyboard) -| (adsr1);
\draw[thick,->](keyboard) -| (adsr2);
\draw[thick,->](keyboard) |- (vco1);
\draw[thick,->](keyboard) |- (vco2);
\draw[thick,->](noise) to [out=0,in=170](vcf);
\draw[thick,->](vcf) -- (vca);
\draw[thick,->](vca) -- (output);
\draw[thick,->](offset) to [out=0,in=170](vco2);
```

Listing 5: Code for Figure 6 (extract)

3 Conclusion

TikZ never ceases to amaze when it comes to creating graphics. It would probably have been much faster in a drawing program, but it was a lot of fun with TikZ.

◇ Uwe Ziegenhagen
Escher Str. 221
50739 Cologne
Germany
ziegenhagen (at) gmail dot com
https://www.uweziegenhagen.de

Diagrams with TikZ: A practical example, part II

Uwe Ziegenhagen

Abstract

In the first part of this article I showed how you can draw a block diagram with TikZ. In this article, I will focus on improvements and alternative approaches.

1 Our starting point

Figure 1 shows the starting point for today's article, the final result of the first article on this topic. If you compare the illustration with the original graphic, also printed in the last article, you will notice (here marked with red circles; all figures have been scaled for *TUGboat*, and color is grayscaled in print):

- The arrow from `offset` to `VCO2` should be straight, not slightly curved at the arrowhead.
- The arrows from `Noise` and `VCO1` to the `VCF` node are not curves in the original, but also angular lines.

Of course, one could argue that the graphics were already "good enough", but then one could just use Word. :-)

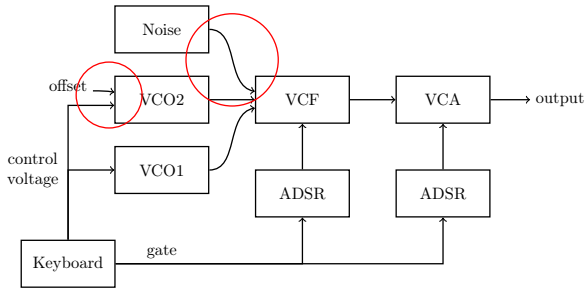


Figure 1: The result of the first article with two "issues" highlighted

In the last article, the positions of the individual nodes were explicitly determined using coordinates. This approach is completely legitimate, but TikZ also supports the positioning of nodes via their relative distances to other nodes. In this article I would like to show how this works and how we can convert the curves mentioned above into attractive lines.

If you want to replicate what I do, you can find the source code in Listing 1. It is also available via my website.[1]

```
\begin{tikzpicture}[
box/.style={rectangle,thick,draw=black,
minimum width=20mm,
minimum height=10mm,align=center}]

\node at (0,1) [box] (keyboard) {Keyboard};

\node at (2,6) [box] (noise) {Noise};
\node at (2,4.5) [box] (vco2) {VCO2};
\node at (2,3) [box] (vco1) {VCO1};

\node at (5,4.5) [box] (vcf) {VCF};
\node at (5,2.5) [box] (adsr1) {ADSR};

\node at (8,4.5) [box] (vca) {VCA};
\node at (8,2.5) [box] (adsr2) {ADSR};

\node at (10.5,4.5) (output) {output};
\node at (0,4.81) (offset) {offset};
\node at (2,1.25) (gate) {gate};
\node [align=center] at (-0.75,3) (cv)
{ control \\ voltage };
\draw[thick,->] (vco1) to [out=0,in=190](vcf);
\draw[thick,->] (vco2) -- (vcf);
\draw[thick,->] (adsr1) -- (vcf);
\draw[thick,->] (adsr2) -- (vca);
\draw[thick,->] (keyboard) -| (adsr1);
\draw[thick,->] (keyboard) -| (adsr2);
\draw[thick,->] (keyboard) |- (vco1);
\draw[thick,->] (keyboard) |- (vco2.base west);
\draw[thick,->] (noise) to [out=0,in=170](vcf);
\draw[thick,->] (vcf) -- (vca);
\draw[thick,->] (vca) -- (output);
\draw[thick,->] (offset.base east) to
                    [out=0,in=171] (vco2);
\end{tikzpicture}
```

Listing 1: Code for Figure 1

2 Fundamentals of positioning nodes

Let us first take care of the alternative positioning of the nodes before we turn to the coordinate calculations. The following example was borrowed from a `tex.sx` article by Gonzalo Medina.[2] It shows how you can easily position individual nodes relative to each other using the TikZ `positioning` library.

We start again with a simple node, which we set at the coordinates (0,0). Using information like `below = of Nodename` we can instruct TikZ to position the new nodes. In addition to "left", "right", "above" and "below" there are also "left" and "right" for "above" and "below" variants. Figure 2 shows the corresponding positions.

```
\begin{tikzpicture}[
box/.style={rectangle,thick,draw=black,
minimum width=20mm, minimum height=10mm,
```

[1] `www.uweziegenhagen.de`

[2] `tex.stackexchange.com/questions/69439/how-can-i-achieve-relative-positioning-in-tikz`

```
align=center}]
\node at (0,0) [box] (a) {a};
\node [below = of a,box] (b) {below};
\node [above = of a,box] (c) {above};
\node [left = of a,box] (d) {left};
\node [right = of a,box] (e) {right};
\node [below left = of a,box] (f) {below left};
\node [below right= of a,box] (g) {below right};
\node [above left = of a,box] (h) {above left};
\node [above right= of a,box] (i) {above right};
\end{tikzpicture}
```

Listing 2: Code for Figure 2

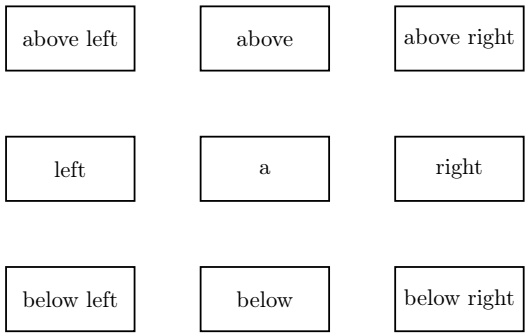


Figure 2: Relative node positioning with the `positioning` library

In addition to these relative specifications, numerical adjustments can be made, as the example in Listing 3 and Figure 3 shows.

Here the labels `A1` to `A4` are placed not only to the right of `a`, but at a defined distance of one to four centimeters.

This even works for the X and Y directions separately; see the labels `A5` and `A6`. `A5` corresponds to "below right", while for `A6` the `below` distance was set to two centimeters and the `right` distance to one centimeter. The specification `below right=2cm and 1cm of a` does not simply correspond to the X and Y coordinates; here it is more of a (Y,X) pair.

```
\begin{tikzpicture}[
box/.style={rectangle,thick,draw=black,
minimum width=20mm, minimum height=10mm,
align=center}]
\node at (0,0) [box] (a) {a};
\node (A1) [right=1cm of a] {A1};
\node (A2) [right=2cm of a] {A2};
\node (A3) [right=3cm of a] {A3};
\node (A4) [right=4cm of a] {A4};
\node (A5) [below right=0cm and 0cm of a] {A5};
\node (A6) [below right=2cm and 1cm of a] {A6};
\end{tikzpicture}
```

Listing 3: Code for Figure 3

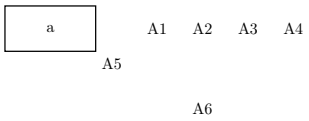


Figure 3: Numerical offsets with node positioning with the `positioning` library

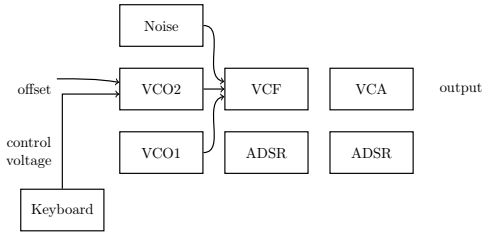


Figure 4: All nodes positioned relatively

We now have all the information we need to define our example synthesizer nodes relatively. I have also drawn the arrows necessary for further adjustments; the source code can be found in Listing 4, the output in Figure 4.

```
\begin{tikzpicture}[
box/.style={rectangle,thick,draw=black,
minimum width=20mm, minimum height=10mm,
align=center,node distance=0.5cm}]

\node at (0,0) [box] (noise) {Noise};
\node[box, below = of noise] (vco2) {VCO2};
\node[box, below = of vco2] (vco1) {VCO1};

\node[box, right = of vco2] (vcf) {VCF};
\node[box, right = of vcf] (vca) {VCA};

\node[box, below = of vcf] (adsr1) {ADSR};
\node[box, below = of vca] (adsr2) {ADSR};

\node[right = 0.5cm of vca] (output) {output};
\node[box, below left = 0.5cm of vco1]
        (keyboard) {Keyboard};
\node[left = 1.5cm of vco2](cv) {offset};

% \parbox not needed
\node[left = 1.5cm of vco1,align=center](cv)
{control \\ voltage};

\draw[thick,->] (keyboard) |- (vco2.base west);
\draw[thick,->] (vco1) to [out=0,in=190] (vcf);
\draw[thick,->] (vco2) -- (vcf);
\draw[thick,->] (noise) to [out=0,in=170] (vcf);
\draw[thick,->] (offset.base east) to
                    [out=0,in=171] (vco2);
\end{tikzpicture}
```

Listing 4: Code for Figure 4

3 Calculations

3.1 The arrow from `offset` to `VCO2`

Having implemented the relative positioning of the nodes, next is calculating the points through which we want to draw our lines. Figure 5 shows this:

- the left edge of `VCO2` has the entry point for the arrow of `offset`, marked with a red '×' (grayscaled for print).
- blue dots mark the arrow path from `Noise` to `VCF`
- red asterisks mark the arrow path from `VCO1` to `VCF`

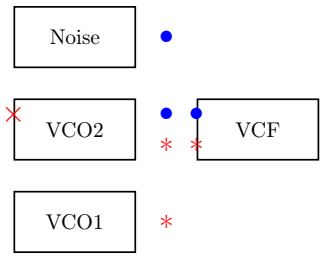


Figure 5: Points to be calculated

For coordinate calculations, the `calc` library — which is loaded via `\usetikzlibrary{calc}` — offers various calculation commands, but only one syntax is relevant for us:

`($`⟨*coordinate*⟩`!`⟨*number*⟩`!`⟨*coordinate*⟩`$)`

⟨*coordinate*⟩ stands for a coordinate which — perhaps explained somewhat simply — is a node without the (optional) text. ⟨*number*⟩ is a number between 0 and 1 and indicates, in decimal notation, the percentage by which we move from coordinate 1 to coordinate 2. Thus, `0.25` stands for a quarter of the way, `0.5` for the midpoint between the two coordinates, etc.

So now we can determine the center points between two nodes/coordinates. Figure 7, taken from an article by TeXWelt,[3] shows the relevant anchors in a rectangle, which we will now work with.

We can determine the entry point of the `offset` arrow directly: it is the coordinate that lies exactly in the middle between `north west` and `west`. Listing 5 shows the source code for the calculation, Figure 6 shows the corresponding output.

```
\node at (0,0) [box] (noise) {Noise};
\node [box, below = of noise] (vco2) {VCO2};
\coordinate (coordoffset) at
     ($(vco2.west)!0.5!(vco2.north west)$);
```

[3] `texwelt.de/fragen/21486/tikz-wie-muss-ich-den-anchor-rechte-stellen`

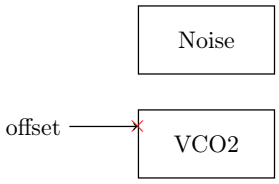


Figure 6: Output of Listing 5

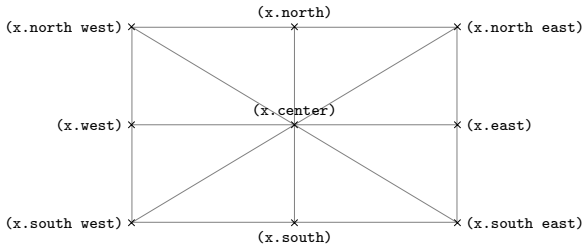


Figure 7: Anchors in a rectangle node, derived from `texwelt.de/fragen/21486/tikz-wie-muss-ich-den-anchor-richtig-setzen`

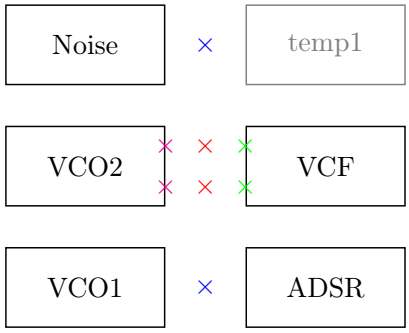


Figure 8: Temporary node `temp1` and the points which the lines will go through

```
\node at (coordoffset)
          {\textcolor{red}{$\times$}};

\node [left = of coordoffset](offset) {offset};
\draw [thick,->] (offset) -- (coordoffset);
```

Listing 5: Code for Figure 6

3.2 The path of the arrow

Determining the points for the arrow paths is a little more complex: First we define a node `temp1` that is to the right of `Noise`. It will not appear in the final diagram, but is used only to calculate the correct distances for the top arrow. For the arrow from `VCO1` to `VCF` we do not need such an auxiliary node; here we simply use the left of the two `ADSR` nodes. Figure 8 shows them along with the auxiliary coordinates for the lines.

The midpoints between `Noise` and `temp1` or between `VCO1` and `adsr1` then form the first coordinates for the arrow paths; in the source code they are called `noisetemp1` and `vco1adsr1`.

Next, we determine the auxiliary coordinates for the nodes `VCO2` and `VCF`, which lie between the middle and upper corner or middle and lower corner, respectively, a total of four. These coordinates only have the purpose of being auxiliary coordinates for the final calculation; we will hide them in the final graphic.

We now have all the points and auxiliary points to draw the final diagram in Figure 9; Listing 6 contains the complete code.

```
\begin{tikzpicture}[box/.style={rectangle,thick,
draw=black,minimum width=20mm,
minimum height=10mm,
align=center,node distance=0.5cm}]

\node at (0,0) [box] (noise) {Noise};
\node [box, below = of noise] (vco2) {VCO2};
\node [box, below = of vco2] (vco1) {VCO1};
\node [box, right = 1cm of vco2] (vcf) {VCF};
\draw [thick,->] (vco2) -- (vcf);

\coordinate (coordoffset) at
    ($(vco2.west)!0.5!(vco2.north west)$);

\node [left = of coordoffset](offset) {offset};
\draw [thick,->] (offset) -- (coordoffset);
\node [right = 1cm of noise] (temp1) {};

\coordinate (vcf1) at
     ($(vcf.west)!0.5!(vcf.north west)$);
\coordinate (vcf2) at
     ($(vcf.west)!0.5!(vcf.south west)$);
\coordinate (vco21) at
     ($(vco2.north east)!0.5!(vco2.east)$);
\coordinate (vco22) at
     ($(vco2.south east)!0.5!(vco2.east)$);

\node [box, right = of vcf] (vca) {VCA};
\node [box, below = of vcf] (adsr1) {ADSR};
\node [box, below = of vca] (adsr2) {ADSR};
\node [right = 0.5cm of vca] (output) {output};
\node [box, below left = 0.5cm of vco1]
       (keyboard) {Keyboard};

\node[left = 1.5cm of vco1](cv)
     {control \\ voltage};

\coordinate (noisetemp1) at
     ($(noise.east)!0.5!(temp1.west)$);
```

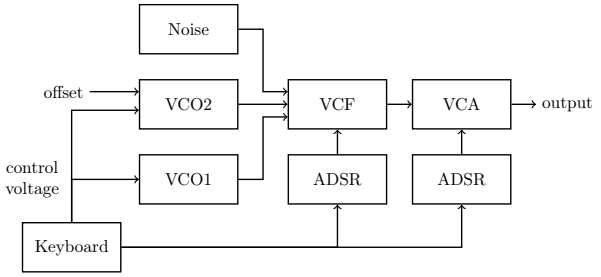


Figure 9: Our final diagram

```
\coordinate (vco1adsr1) at
     ($(vco1.east)!0.5!(adsr1.west)$);

\coordinate (vco2vcf1) at
     ($(vco21)!0.5!(vcf1)$);
\coordinate (vco2vcf2) at
     ($(vco22)!0.5!(vcf2)$);

\draw[->,thick] (vco1.east) -- (vco1adsr1)
    --  (vco2vcf2) -- (vcf2);
\draw[->,thick] (noise.east) -- (noisetemp1)
    --  (vco2vcf1) -- (vcf1);

\draw[->,thick] (adsr1) -- (vcf);
\draw[->,thick] (adsr2) -- (vca);
\draw[->,thick] (vcf) -- (vca);
\draw[->,thick] (vca) -- (output);

\draw [thick,->] (keyboard) -| (adsr1);
\draw [thick,->] (keyboard) -| (adsr2);
\draw [thick,->] (keyboard) |- (vco1);
\draw [thick,->] (keyboard) |- (vco2.base west);
\end{tikzpicture}
```

Listing 6: Code for Figure 9

4 Conclusion

Was the result worth that much effort? Yes, for several reasons. First of all the result is just as I wanted it. With Affinity Designer I would have been much quicker, but with TikZ there is the learning curve: in case I ever need a similar image again I can re-use parts of this code. And finally, it was a nice example to learn about some of the tools TikZ offers.

◇ Uwe Ziegenhagen
Escher Str. 221
50739 Cologne
Germany
ziegenhagen (at) gmail dot com
https://www.uweziegenhagen.de

Asemic writing using MetaFun in ConTEXt

Keith McKay

Abstract

The article describes ways to generate asemic writing using MetaPost, MetaFun, and LuaMetaFun in ConTEXt. The term asemic means no semantic content and thus asemic writing is writing without semantic content or meaning. It has a thriving community of artists producing many different styles which can best be described as a hybrid form of abstract art. Examples of hand-written and ConTEXt generated asemic writing will be presented along with examples of the code used to produce them.

About five years ago I stumbled across a hybrid form of abstract art known as *asemic writing*. Although I'm not an artist I found the works very appealing and as I looked further into the term asemic, I became even more intrigued when I found that it means something with *no semantic content* or, in layman's terms, *something without meaning*. The writing looks like it has meaning since in many cases it contains gestural lines connecting 'characters' but on closer inspection it has no meaning at all.

Asemic writing was given that name by two visual poets, Tim Gaze and Jim Leftwich, in the late nineties, although it can be argued that it has been around much longer.[1] Chinese characters and Islamic calligraphy to some western eyes may be thought of asemic but not by the cultures who wrote it. Vice versa, western Latin script could be thought of as asemic by those cultures. Going back even further, what are we to make of the cup and ring marks of Neolithic rock art? We don't know what they mean today but they must have had meaning to the people in those times.

Modern day asemic writing comprises a multitude of styles,[2] but my interest was in handwritten asemics because the randomness in the style appealed to me.

In the figures below I show some examples of asemic writing. Figures 1 and 2 show the variation of styles of asemic writing.[3] Figure 3 is one of my own pieces of asemic hand writing.[4]

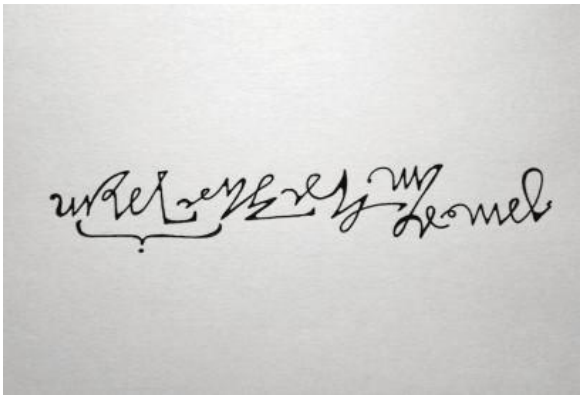


Figure 1: Asemic writing by Geof Huth.

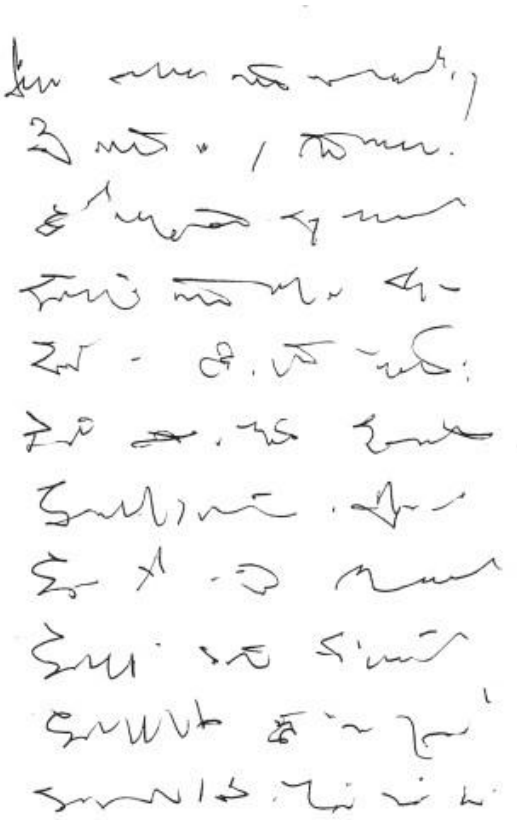


Figure 2: Asemic writing by Jim Leftwich.

At about the same time as I found asemic writing I was reading the MetaFun manual, and I thought maybe I could generate asemic writing using MetaPost and MetaFun code in ConTEXt. After all, the asemic writing looked like a path to which randomness had been applied and then drawn on the page with a pen. MetaFun has a randomized function but it wasn't what I was looking for so I set upon writing one, which was also a good way for me to start to learn some MetaFun coding. My first attempts were,

[1] See *Asemic: the Art of Writing* by Peter Schwenger, University of Minnesota Press, 2019.

[2] See *An Anthology of Asemic Handwriting* edited by Tim Gaze and Michael Jacobson, 2013. It can be downloaded from the Internet Archive.

[3] Both of these examples are taken from *An Anthology of Asemic Handwriting*.

[4] This piece was exhibited at: GRAPHOS — MUESTRA INTERNATIONAL DI ESCRITURA ASÉMICA, 01–18 Nov. 2023, Buenos Aires, Argentina.

doi.org/10.47397/tb/45-3/tb141mckay-asemic

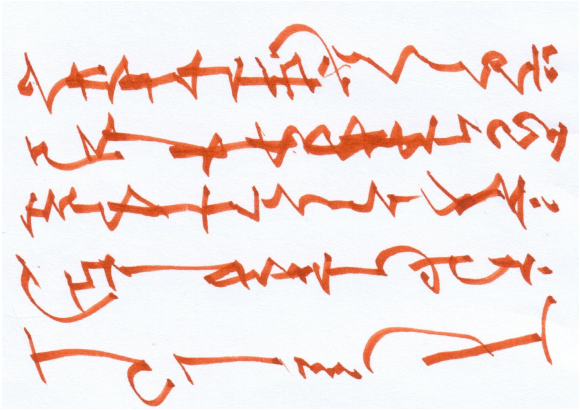


Figure 3: Asemic writing by Keith McKay.

to say the least, a bit clunky, although they worked and I was able to improve them with time.

My logic was as follows. Create a path of a certain length and decide how many points, i.e 'characters', it should contain. Iterate along the path and at each point add a small amount of randomness to the xpart and ypart of the point. Using these new points create a new asemic path.

To implement this I created two vardefs: One to generate a random number within a range of values, usually something like −0.1 to +0.1, and another to create the asemic path as described above.

```
vardef randnumRange(expr mini,maxi) =
  randnum := ((maxi - mini) * uniformdeviate(1))
             + mini;
  randnum
enddef;
```

The asemic path generator vardef has four variables: a path, the number of points in that path and the minimum and maximum for the range of a random number. The last two variables are used within the `vardef randnumRange()`.

```
vardef Asemic(expr aPath,nPoints,minii,maxii) =
path asemicPath;
mini := minii; maxi := maxii; nbPoints := nPoints;
asemicPath :=
  ((((xpart point 0cm on aPath)/72)*2.54)*cm,
   (((ypart point 0cm on aPath)/72)*2.54)*cm);
for x = 1 upto nbPoints:
  ycoord := ((ypart point(x/nbPoints) along aPath)/72)
            *2.54 + randnumRange(mini,maxi);
  xcoord := ((xpart point(x/nbPoints) along aPath)/72)
            *2.54 + randnumRange(mini,maxi);
  asemicPath := asemicPath ...
                {curl 100}(xcoord*cm, ycoord*cm);
endfor;
if ((point 0 along aPath)
    = (point nbPoints along aPath)):
  asemicPath := asemicPath ... cycle;
fi;
asemicPath % return
enddef;
```

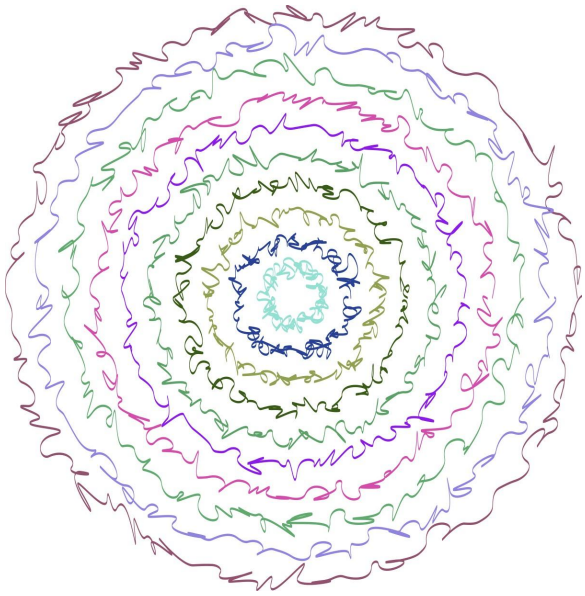


Figure 4: Concentric asemic circles.

In addition to the vardefs which created the asemic line, I wrote a small def to create a calligraphic styled pen with a randomly generated rgb colour.

```
def coloredPath(expr s, t) =
  withpen pencircle xscaled s yscaled t
    withcolor (uniformdeviate(1),
               uniformdeviate(1),uniformdeviate(1));
enddef;
```

Using these three macros I could create a number of asemic lines on a page with randomly generated colours, like so:

```
\startMPpage
path aPath;
numeric nPoints, mini, maxi;
nPoints := 100;
mini := -0.125; maxi := 0.125;
for i = 10 downto 1:
  aPath := (0cm, i*cm) -- (10cm, i*cm);
  draw Asemic(aPath,100,-0.125,0.125)
       coloredPath(.1mm,.4mm);
endfor;
\stopMPpage
```

Lines of text can get boring, so replacing `aPath` with `aPath:= fullcircle scaled(i*cm);` gives a series of randomly coloured asemic concentric circles, as shown in Figure 4. Indeed any path, open or closed, can be made asemic using our vardefs `Asemic` and `randnumRange`.

But we don't need to stop here. A page of text consists of lines of words made up of characters, and, these words are separated with spaces, so how can we use the asemic vardefs to create lines of asemic words? This was a little tricky to solve at first because there are a number of things to keep in mind. We have already decided on a path which has a certain number of points; we decide on the number of spaces between

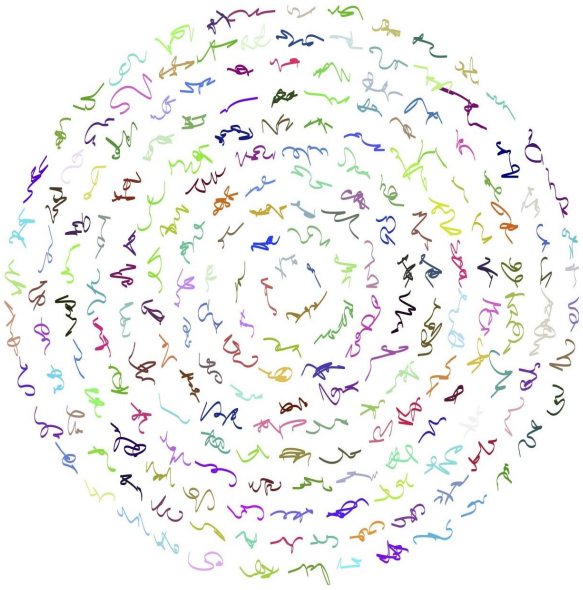


Figure 5: Concentric circles of asemic words.

the words and we can randomly generate the number of characters for each word. However, we must make sure that the last word does not overshoot the end of the line and to do this we must keep a running total of the number of points used along the path. When we get to a word that does overshoot the end of the line we truncate it so that it fits, thus giving the impression of justified text.

```
\startMPinclusions
% Put vardef randnumRange() here.
% Put vardef Asemic() here.
% Put def coloredPath()here
\stopMPinclusions
\startMPpage
path myPatha, myPathb, myPathc, myPathd, asubPath;
numeric nPoints, nchars, mini, maxi, minword, maxword,
        spaces, beginPath, endPath;
boolean ending;
nchars := 400; minword := 5; maxword := 10;
mini := -0.125; maxi := 0.125; spaces := 3;
for i = 10 downto 1:
  myPatha := (0cm, i*cm) -- (10cm, i*cm);
  myPathb := fullcircle scaled(i*cm)shifted(16cm,5cm);
  myPathc := fullsquare scaled(i*cm)shifted(5cm,16cm);
  myPathd := fulltriangle scaled(i*cm)shifted(15cm,
                                              16cm);
  nPoints := (nchars * (i/10));
  for j = myPatha, myPathb, myPathc, myPathd:
  ending := false;
  wordlength := round(randnumRange(minword,maxword));
  beginPath := 1;
  endPath := beginPath + wordlength;
  forever:
    if (nPoints - beginPath) < 18:
      ending := true;
      endPath := nPoints - 1;
      t := (beginPath/nPoints) * length j;
      u := (endPath/nPoints) * length j;
      asubPath := subpath(t,u) of j;
      wordlength := (nPoints - 1) - beginPath;
      draw Asemic(asubPath,wordlength,mini,maxi)
           coloredPath(.1mm,.4mm);
    else:
      ending := false;
      t := (beginPath/nPoints) * length j;
      u := (endPath/nPoints) * length j;
      asubPath := subpath(t,u)  of j;
      draw Asemic(asubPath,wordlength,mini,maxi)
          coloredPath(.1mm,.4mm);
      beginPath := endPath + spaces;
      endPath := beginPath + wordlength;
      wordlength := round(randnumRange(minword,
                                       maxword));
    fi;
    exitif ending = true;
  endfor;
  endfor;
endfor;
\stopMPpage
```

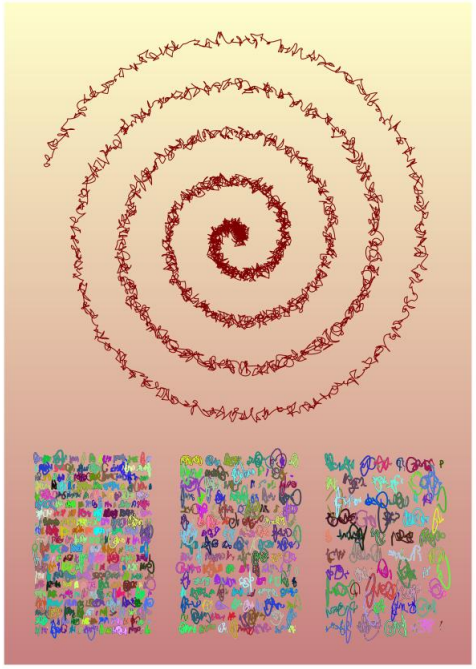


Figure 6: Asemic spiral with asemic writing.

Figure 5 shows concentric asemic circles consisting of randomly sized words with the density of points on the original circle adjusted for diameter. This is one of the asemic shapes produced in the above code. Readers are welcome to try it out; you'll need a ConTEXt distribution installed (`contextgarden.net`).

We now have the basic building blocks for pages of computer generated asemic writing. Figure 6 and the following page show some images that I have produced using ConTEXt.

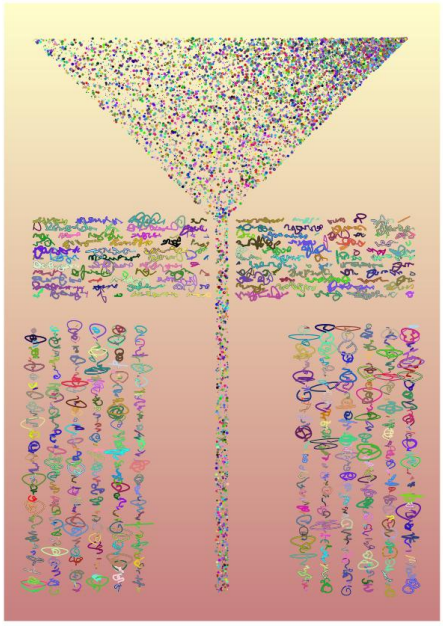


Figure 7: Horizontal and vertical asemic writing.

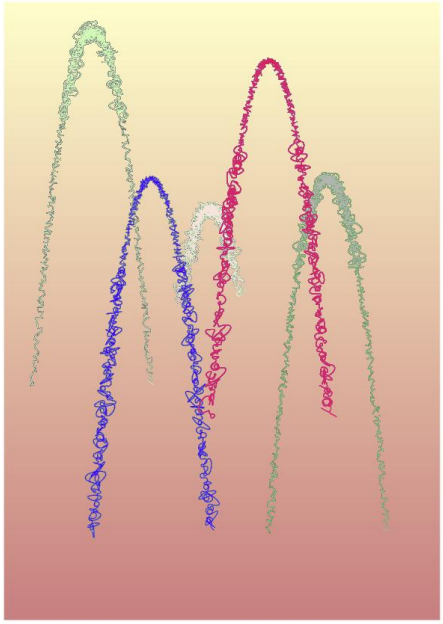


Figure 10: Asemic functions.

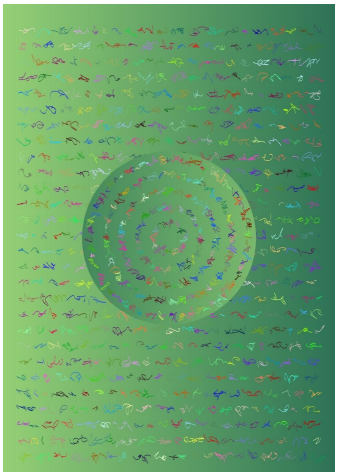


Figure 8: Dimpled Asemic writing.

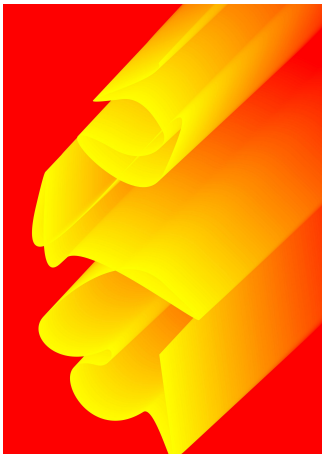


Figure 11: Large blended asemic character.

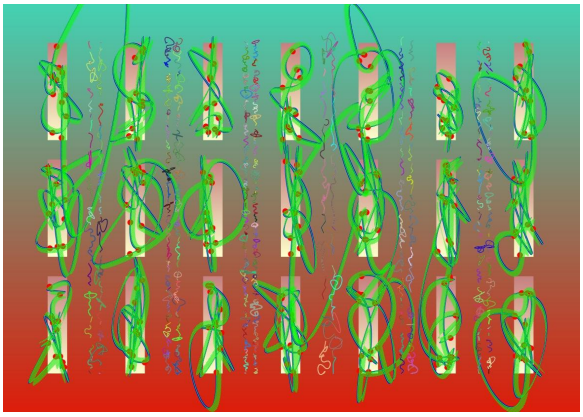


Figure 9: Asemic characters.

Figure 12: Asemic writing.

Recently, Hans Hagen suggested to me some improvements to the code which would speed up the calculation. In the LuaMetaFun manual there is some information about fast loops over paths. Normally if MetaPost has to find a point q on a path, it has to start at 1 and move along the path to find q, but LuaMetaFun has `for i within path : ... pathpoint ... endfor;`. In conjunction with the new primitive `arcpointlist`, this considerably speeds up the computation. The revised code is as follows:

```
vardef Asemic(expr aPath, nPoints, mini, maxi) =
  path aasemicPath; numeric nbPoints;
  nbPoints := nPoints;
  aasemicPath := aPath;
  aasemicPath := arcpointlist nbPoints of aasemicPath;
  aasemicPath :=
    for i within aasemicPath:
      randomiser(pathpoint, mini, maxi)
      .. tension 2.5 ..
    endfor
    nocycle
    ;
  aasemicPath
enddef;
```

with a new `vardef randomiser()` to randomise the point on the path.

```
vardef randomiser(expr p, mini, maxi) =
  (
    xpart p + randnumRange(mini, maxi)*cm,
    ypart p + randnumRange(mini, maxi)*cm
  )
enddef;
```

Along with Mikael Sundqvist, Hans Hagen has also been adding additional functionality to LuaMetaFun in the form of envelopes. I have used these to generate asemic writing. I will not go into detail about envelopes since it is outside my area of expertise, but in simple terms, an envelope is the closed path we get when we run a pen over a path. This may not seem very exciting; however, when used with `withshademethod " "` and `withshadecolors(colora,colorb)` some very interesting effects can be obtained. The LuaMetaFun manual goes into more detail and readers are encouraged to read it.[5]

Here is a code snippet in which I used envelopes to produce the shaded asemic words shown in Figures 13 and 14. The two figures are examples of MetaPost code taken from `tlhiv.org` to which I then added my asemic code.[6] An envelope (`asPathEnvelope`) is created over an asemic path (`asPath`) and two colours are randomly generated which are then used to fill and shade the envelope.

[5] The LuaMetaFun manual is in the ConTEXt distribution.
[6] `tlhiv.org/MetaPost/examples/examples.html`

Figure 13: Trefoil with asemic shaded writing.

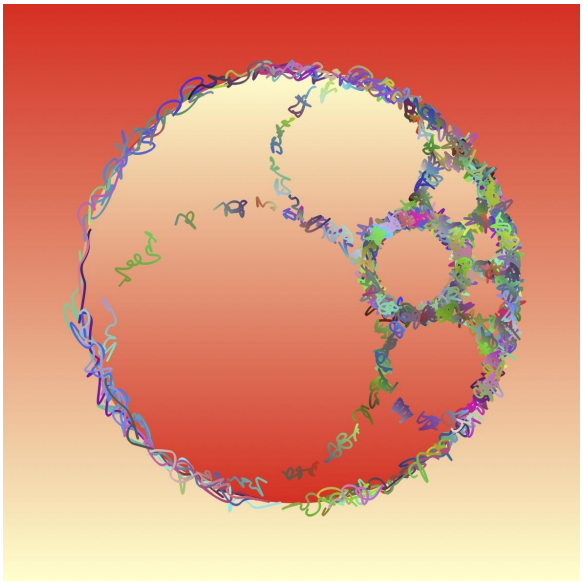


Figure 14: Asemic circles within circles.

```
% snip
asPath := Asemic(aPath, nPoints,mini,maxi);
asPathEnvelope := envelope pensquare scaled 0.3mm
                          rotated 45 of asPath;
definecolor [name = "Mycolor1",
             r = uniformdeviate(1),
             g = uniformdeviate(1),
             b = uniformdeviate(1)];
definecolor [name = "Mycolor2",
             r = uniformdeviate(1),
             g = uniformdeviate(1),
             b = uniformdeviate(1)];
fill asPathEnvelope yscaled 1
        withshademethod "linear"
        withshadecolors ("Mycolor1","Mycolor2");
% snip
```

Finally, I will show some images of the effect of envelopes on a series of paths forming the shape of a spiral with the paths asemic lines, and lines of asemic words.

Figure 15 is coded as follows:

```
pair A,B;
path p, psubPath, asPathEnvelope;
A = (0cm,0cm);B = (5cm,0cm);
p = A{dir 60} .. B;
fill fullsquare scaled 12cm withcolor 0.5[blue,white];
for i = 0 step 10 until 350:
  psubPath := (subpath((0.1 * length p),
                       (1 * length p))
                of p) rotated i;
  asPathEnvelope := envelope pensquare scaled 0.5mm
                    of psubPath;
  fill asPathEnvelope
       withshade define_linear_shade
                 ((cosd(i) * 0.5cm, sind(i) * 0.5cm),
                  (cosd(i) * 5cm, sind(i) * 5cm),
       0.5[blue,white],
       white);
endfor;
```

and Figures 16 and 17 have the asemic vardefs applied to them as shown previously.

This has been a whirlwind tour of asemic writing but I hope it has given the reader a taste of this hybrid art form. As I said at the beginning, I am not an artist and I should add, not a good coder either, but the experience of using ConTEXt with MetaPost, MetaFun and LuaMetaFun in creating asemic writing through code has led me down a new path. In my exploration of generating asemic writing through code, I still ponder if the computer-generated asemic writing is truly asemic since the code to produce it has to be syntactically correct and therefore has meaning. If it is incorrect then there is no output. So, I'll leave you with this thought. Can something with meaning produce something without meaning?

With thanks to Hans Hagen, Mikael Sundqvist, and the ConTEXt mailing list for help in problem solving.

◇ Keith McKay
Maryhill, Glasgow
Scotland, UK
mckaymeister (at) gmail dot com

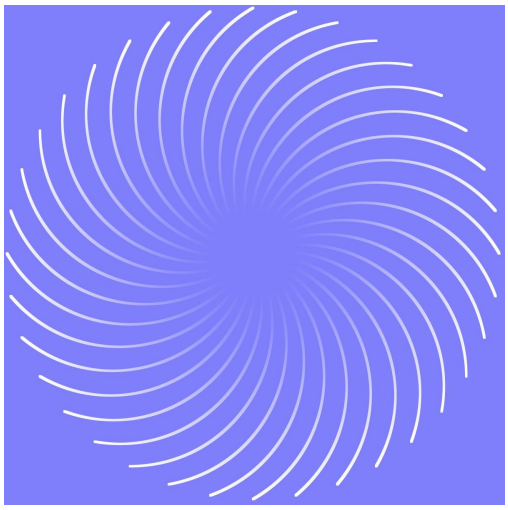


Figure 15: Spiral with envelopes.

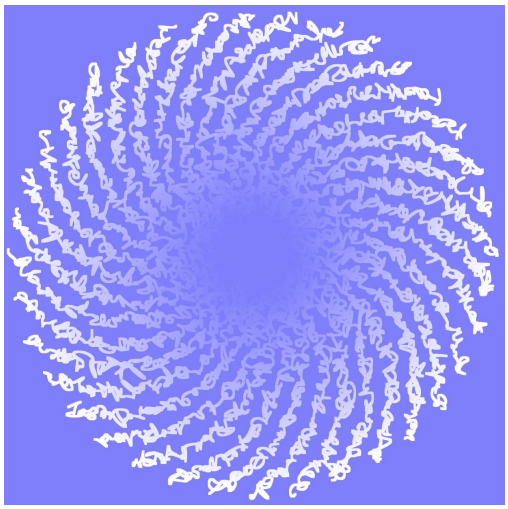


Figure 16: Spiral of asemic lines.

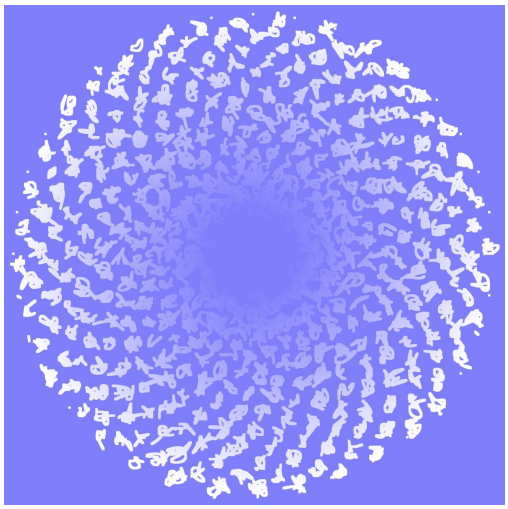


Figure 17: Spiral of lines of asemic words.

Easy periodic tables of elements with pgf-PeriodicTable

Hugo Gomes

Abstract

With the pgf-PeriodicTable package you can obtain a Periodic Table of Elements simply by typing \pgfPT. Then, the package does the *magic* of building the periodic table with the indispensable help of Ti*k*Z. For a customized periodic table, you can use \pgfPT[⟨*options*⟩]. Some of the features of the pgf-PeriodicTable package will be described here, as well as the motivation for its construction.

1 Introduction

The aim of the pgf-PeriodicTable package is to provide the Periodic Table of Elements in a simple way: this can be achieved by just typing \pgfPT (figure 1) inside any document to be processed with pdfLaTeX, LuaLaTeX or XeLaTeX. (See also figure 21 at the end of the article.)

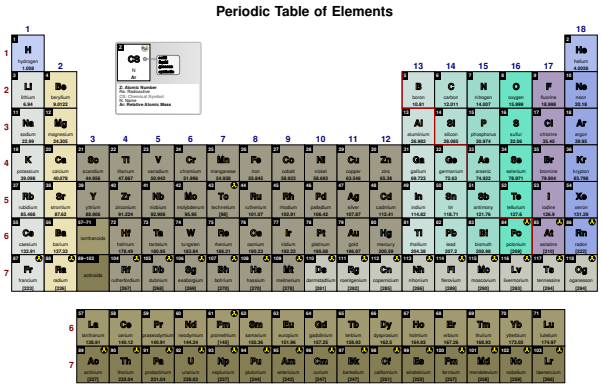


Figure 1: The Periodic Table obtained using \pgfPT.

To use the package it must be loaded by writing in the preamble, as usual in any LaTeX document, \usepackage{pgf-PeriodicTable}. The package ca also be loaded with options to set the default language or to use Devanagari or Mandarin numerals in the atomic number, periods and/or groups.

The default language is English (whose *language flag* is en); also, French (fr), German (de), Portuguese from Portugal (pt) and from Brazil (br), Spanish (es) and Italian (it) are available, all of them built-in, and Dutch (nl) is provided by a *user* translation. To set the default language at package loading use:

\usepackage[⟨*language flag*⟩]
{pgf-PeriodicTable}

for a *built-in* language; or

\usepackage[userlang=⟨*language flag*⟩]
{pgf-PeriodicTable}

for a *user* language.

If desired the default language can be globally changed using \pgfPTsetLanguage{⟨*language flag*⟩} anywhere in the document.

After loading the pgf-PeriodicTable package the main command available is \pgfPT. It can be used with or without options. In the latter case, the options are for customizing the Periodic Table and how its contents will be displayed. The information available, from all the actual 118 elements, is: atomic number, element name, chemical symbol, relative atomic mass, standard relative atomic mass, radioactivity, atomic radius (empirical), covalent radius, ionic radius, first ionization energy, electronegativity (Pauling), electroaffinity, oxidation states, melting point (in Kelvin and Celsius), boiling point (in Kelvin and Celsius), electron distribution, electronic configuration (increasing n and increasing $n + \ell$), density, specific heat capacity, thermal conductivity, lattice structure, lattice constants (a, b, c and c/a ratio), year of discovery, country of discovery and visible range spectral lines.

The options can be divided into two subsets. One of them affects the appearance of the entire Periodic Table, while the other concerns the contents of each cell of the Periodic Table, to be called a *unit cell*. Both follow the \pgfPT command in square brackets, forming a comma separated list of any of the following elements:

- a key or a key=value pair,
- a style or a style=value pair,
- a *pseudo-style* with a proper syntax:
  style={key 1=value 1, ... , key n=value n}
  where none of the *keys* are mandatory.

2 Periodic Table options

The options described below could be used to customize the Periodic Table, as a whole, in various aspects, such as cell width or cell height, whether the title or legend is shown (show title or show legend), which elements are present (Z list), among others. Not all options will be described here; a complete reference can be found in the pgf-PeriodicTable manual available on CTAN.

Before looking at the options, just one more note: the examples shown here make use of a *global style* established by:

```
\pgfPTstyle[Z list={1,...,36},
    show title=false,show legend=false]
```

However, in some of the examples, they will be locally overridden for the benefit of the context.

doi.org/10.47397/tb/45-3/tb141gomes-periodic

2.1 General layout

Z list sets the element list to display in the Periodic Table. It could be either a `name` or a `comma separated` list of atomic numbers.

```
\pgfPT[Z list={1,...,54}]
```

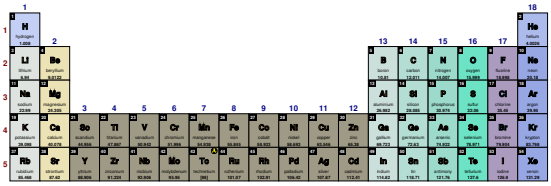


Figure 2: A periodic table with the first five periods.

The `name` can be a built-in name like `all` for all the known elements, `P1` to `P7` for each period, `G1` to `G18` for each group, or a few other predefined names; or any user-defined name via `\pgfPTnewZlist{name}{list}`.

```
\pgfPT[Z list=P4]
```

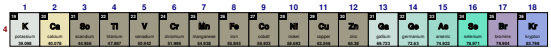


Figure 3: The fourth period of the Periodic Table.

cell width used to set the width of the unit cell of the Periodic Table.

cell height used to set the height of the unit cell of the Periodic Table.

cell size style to set both the width and height of the unit cell.

cell style loads a `named cell style` to use as a layout for the unit cell. The cell style can be one of the built-in styles or it can be defined using the `\pgfPTbuildcellstyle` command.

```
\pgfPT[cell style=pgfPTls,show legend]
```

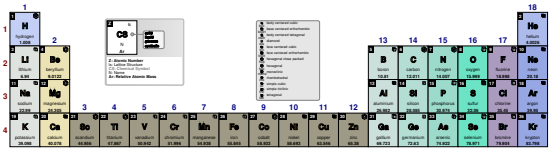


Figure 4: A Periodic Table with a built-in cell style (`pgfPTls`).

```
\pgfPTbuildcellstyle{myT}
  (6,3)% 6 rows by 3 columns
  [(1;1-2;Z),(1;3;radio),
  (2-3;1.5-3.5;CS),(4;1-3;name),
  (5;1-2.5;Tmelt),(5;2.5-3;Tboil),
  (6;1-2.5;TmeltC),(6;2.5-3;TboilC)]
  \pgfPT[cell style=myT,show legend
    Tmelt color=blue!50!black,
    Tboil color=red!50!black,
    TmeltC color=blue,TboilC color=red]
```

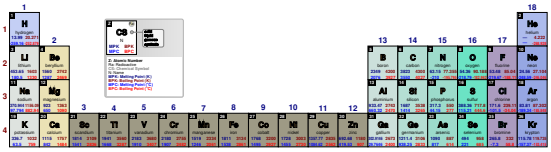


Figure 5: A Periodic Table with a user-defined cell style (myT).

back color scheme loads a `named back color scheme` to use in the Periodic Table. It can be one of the built-in color schemes or one built using the `\pgfPTnewColorScheme` command.

```
\pgfPT[Z list=all,
    back color scheme=pgfPTSoft]
```

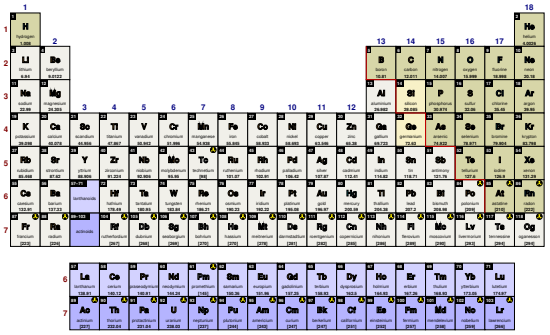


Figure 6: A Periodic Table distinguishing metals, non-metals, silicon and germanium, lanthanoids and actinoids using the built-in `Soft` background color scheme.

IUPAC if true, draws the periodic table with lanthanum and actinium appended to block f; else lanthanum and actinium are shown in group 3.

```
\pgfPT[Z list={P6}]\\
\pgfPT[Z list={La}]
```

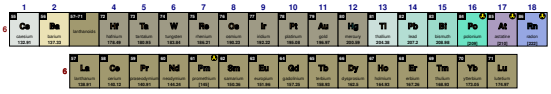


Figure 7: The sixth period with the `IUPAC` option set to `true`.

```
\pgfPT[IUPAC=false,Z list={P6}]\\
\pgfPT[IUPAC=false,Z list={La}]
```

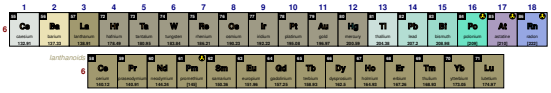


Figure 8: The sixth period with the `IUPAC` option set to `false`.

languages sets a list of languages to display in the Periodic Table. This is a comma-separated list of language flags: `pt`, `en`, `fr`, `de`, `it`, `es` or `br`. All are shown. If a user language has been loaded, the corresponding `ISO 639-1` code can also be used as a language flag; for now just `nl`. *This key locally overrides the default language.*

```
\pgfPT[show title,show legend,
    cell style=pgfPT2lang,
    languages={de,fr}]
```

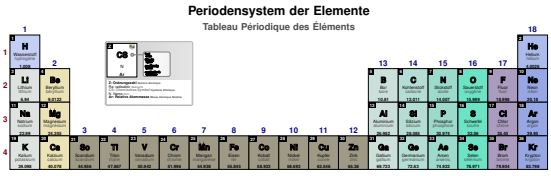


Figure 9: A Periodic Table with two languages, neither of them the default language.

2.2 Title and legend

The title is displayed centered at the top of the Periodic Table. The legend consists of a cell using acronyms for its contents and the corresponding descriptions below that cell or just one cell with the descriptions. It is shown to the right of group 3 and above periods 4 to 6. There is an extra legend shown when the lattice structure or the country of discovery is present in the unit cell of the Periodic Table.

The title or legend can be shown or hidden by setting the `show title` or `show legend` options to `true` or `false`, respectively. The font and color of title can be customized with the options `title font` and `title color`, while the fonts and colors used in the legend descriptions are the same as those used by the respective contents. Exceptions occur for radioactivity, which does not have font and color options, so it is possible to use the `legend radio color` option to set its color in the legend description; for the atomic number, the `legend Z color` option is available to override its previous color; and for the chemical symbol, the `legend CS color` option is available to set its color in the legend description.

```
\pgfPT[show legend,legend acronyms=false,
  legend CS color=red,cell style=pgfPTdisc]
```

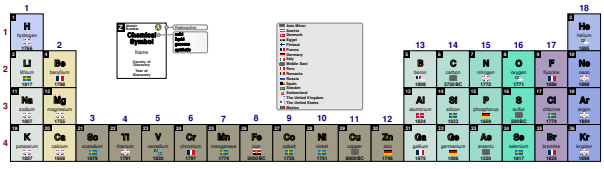


Figure 10: Custom legend of the Periodic Table.

2.3 Periods and groups

By default, period and group numbers are displayed. They can be disabled by setting the `show period numbers` or `show group numbers` options to `false`. The color of the numbers is customized by using the `period label color` and `group label color` options, while the font, for both, can be set using the `label font` option.

To change the group numbering system, the `group numbers` option can be used:

arabic group numbers are shown in arabic numerals as recommended by IUPAC since 1988.

CAS group numbers are shown in Roman numerals and an 'A' or 'B' suffix. This is an older naming scheme, used by the Chemical Abstract Service (CAS), more popular in the United States.

IUPAC group numbers are shown in Roman numerals and an 'A' or 'B' suffix. This is an older naming scheme, used by IUPAC before 1988, more popular in Europe.

CAS* combines the option `CAS` and `arabic`. Roman numerals and 'A' or 'B' suffix are above the group and the arabic numerals above them.

IUPAC* combines the option `IUPAC` and `arabic`. Roman numerals and 'A' or 'B' suffix are above the group and the arabic numerals above them.

```
\pgfPT[group numbers=CAS]
```

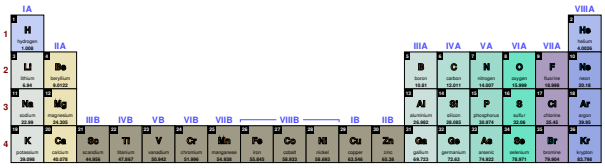


Figure 11: A Periodic Table with the `CAS` numbering system.

The color of the roman numerals can be customized by the `Roman label color` option.

```
\pgfPT[group numbers=IUPAC*,
    Roman label color=teal]
```

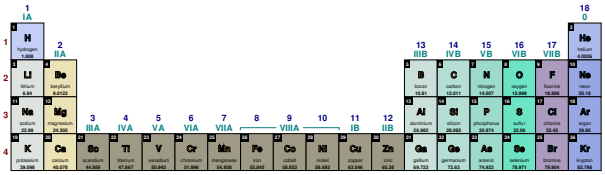


Figure 12: A Periodic Table with `IUPAC` (with the color set to `teal`) and `arabic` numbering systems.

2.4 Blocks and families

With the option `show blocks` the *s*, *p*, *d* and *f* blocks are drawn overlaying the Periodic Table and their labels are shown. Similarly, with the option `show families` the main families — representative elements, transition metals and internal transition metals — are drawn overlaying the Periodic Table and their labels are shown. *Blocks or families are shown only when the* `Z list` *contains, at least, all elements of blocks s, p and d.*

```
\pgfPT[show blocks,Z list=spd]
```

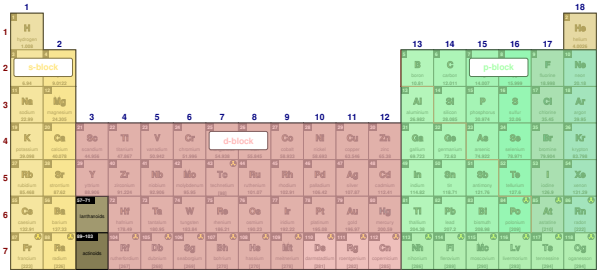


Figure 13: The *s*, *p* and *d* blocks of the Periodic Table.

```
\pgfPT[show families,Z list=all]
```

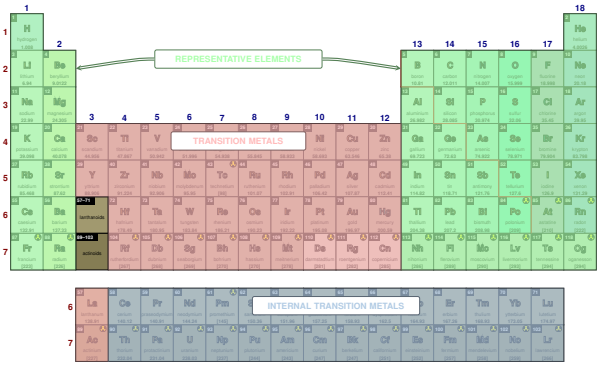


Figure 14: The families of the Periodic Table.

The colors, fonts and line widths used in blocks or families are customizable via the ⟨*item*⟩ `color`, ⟨*item*⟩ `font color` and ⟨*item*⟩ `line width` options, where the ⟨*item*⟩ can be `s block`, `p block`, `d block` or `f block` for the corresponding blocks, or `r family`, `tm family` or `itm family` for the representative elements, transition metals and internal transition metals, respectively.

2.5 Periodic variations

The atomic radius, ionization energy and/or electron affinity trends can be shown with two arrows, a horizontal one placed at the top of the Periodic Table and another one vertically to the left of the Periodic Table. They express the variation over the period and the variation over the group, respectively. To get them use the `show periodic variations` option; keep in mind that they are shown only when the unit cell of the Periodic Table contains the atomic radius, ionization energy and/or electron affinity. If none of these is present, setting the `show periodic variations` option will have no effect.

```
\pgfPT[show periodic variations,
    show legend,Z list=spd,
    cell style=pgfPTR]
```

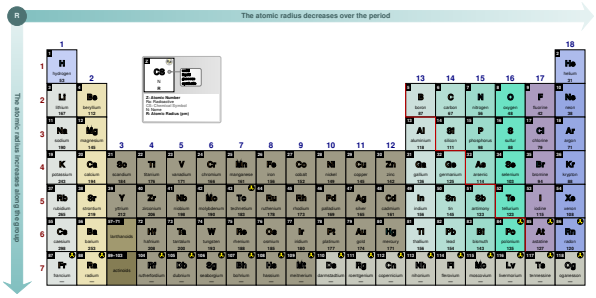


Figure 15: The trend of atomic radius in the Periodic Table.

The color of the arrows and the font and color of the text that expresses the variation work in a similar way to what was mentioned above for blocks and families (Section 2.4). Use the ⟨*item*⟩ `color`, ⟨*item*⟩ `font` and ⟨*item*⟩ `font color` options to configure them. The ⟨*item*⟩ can be, respectively, `R`, `Ei` or `eaff` for the *atomic radius*, *ionization energy* or *electron affinity*.

```
\pgfPT[show periodic variations,
    show legend,Z list=all,
    cell style=pgfPTREi,R color=purple,
    Ei color=colorvariations!50!black,
    varR color=purple]
```

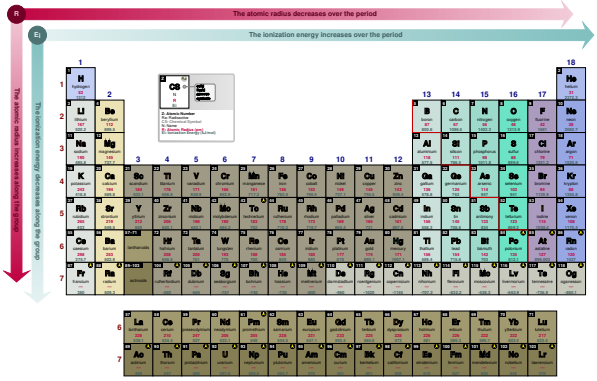


Figure 16: The trends in atomic radius and ionization energy in the Periodic Table.

2.6 Dark mode

A style is available to change the overall appearance of the Periodic Table to a dark mode suitable for on-screen viewing: `\pgfPT[dark mode]`.

```
\begingroup% style reset within this group
\pgfPTresetstyle\pgfPT[dark mode]
\endgroup
```

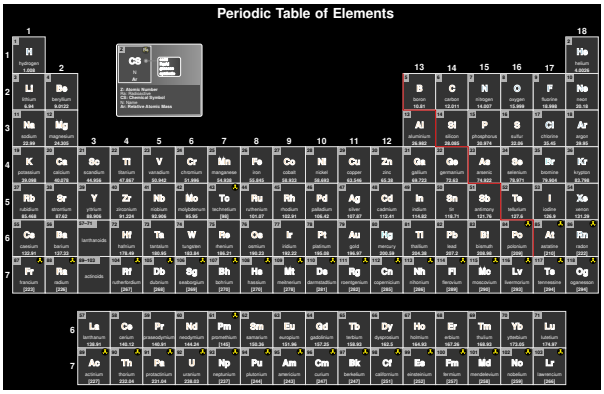


Figure 17: The Periodic Table in `dark mode`.

2.7 Exercise layout

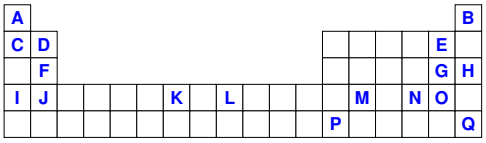
The exercise layout of the Periodic Table provides a special mode where the structure of the Periodic Table is drawn and if desired the period and/or group numbers, the atomic number or some of the elements represented by letters. In the exercise below the Periodic Table is built with:

```
\pgfPT[Z exercise list={1,2,3,4,9,12,17,18,
          19,20,25,27,32,34,35,49,54},
      Z list={1,...,54},cell size=1.5em,
      ex={c=blue,f=\bfseries}]
```

EXERCISE:

In the following scheme of the Periodic Table, the positions of some chemical elements are represented by letters:

THE LETTERS DO NOT CORRESPOND TO THE CHEMICAL SYMBOLS OF THE ELEMENTS.



Using the letters shown:

1. identify the elements of group 2.
2. identify the elements of the 2nd period.
3. ...

Figure 18: An example exercise using the *exercise layout* of the Periodic Table.

The `ex` is a *pseudo-style* available to set the `exercise list font`, `exercise list color` and `exercise list in capitals` options, using shorter syntax,

`ex={f=`⟨*font specs*⟩`,c=`⟨*color*⟩`,caps=`⟨*true/false*⟩`}`

where all of the keys, `f`, `c` and `caps`, are optional.

3 Cell contents options

The unit cell of the Periodic Table is flexible, containing any of the following contents:

Z atomic number.
CS chemical symbol.
name element name.
Ar atomic weight (relative atomic mass).
Ar* standard relative atomic mass.
Ra radioactivity.
R atomic radius.
Rcov covalent radius.
Rion ionic radius.
Ei first ionization energy.
eneg electronegativity (Pauling).
eaff electroaffinity.
O oxidation states.
Tmelt melting point (Kelvin).
TmeltC melting point (degrees Celsius).
Tboil boiling point (Kelvin).
TboilC boiling point (degrees Celsius).
eDist electron distribution.
eConfign electronic configuration (increasing n).
eConfignl electronic configuration (increasing $n + \ell$).
d density.
Cp specific heat capacity.
kT thermal conductivity.
lsa lattice constant: a.
lsb lattice constant: b.
lsc lattice constant: c.
lsca lattice c/a ratio.
DiscY year of discovery.
DiscC country of discovery.
spectra visible range spectral lines.

For each of these content items, two options are available: ⟨*content*⟩ `color` and ⟨*content*⟩ `font`, to set the color and font of the content. The exceptions are *radioactivity*, which does not have font and color options, and *chemical symbol*, which does not have the color option.

For the atomic weight, density and lattice structure constants, the precision of their values can be

defined using the `Ar precision`, `d precision` and `ls precision` options, respectively. For the density and lattice structure constants, the `d unit` and `ls unit` options can be used to set their units. The separator character between energy levels in the electron distribution can be specified with the `eDist sep` option.

A unit cell, by itself and for later use, can be built using the `\pgfPTbuildcell` command. For example, the default unit cell is internally constructed with:

```
\pgfPTbuildcell(5,3)[% 5 lines x 3 columns
      (1;1-2;Z),%line 1: Z in cols 1 and 2
      (1;3;radio),%line 1: radio in col 3
      (2-3;1.5-3.5;CS),%lines 2 and 3: CS from
                       %the middle of col 1 to
                       %the middle of col 3
      (4;1-3;name),%line 4: name in cols 1 to 3
      (5;1-3;Ar)%line 5: Ar in cols 1 to 3
      ]%

\pgfPTpreviewcell[2.5]% scale factor
```

```
Using the last cell built
The build command:
\pgfPTbuildcell(5,3)%
[(1;1-2;Z),(1;3;radio),(2-3;1.5-3.5;CS),(4;1-3;name),
(5;1-3;Ar)]
```

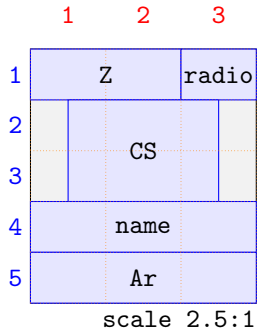


Figure 19: Previewing the default unit cell.

The `\pgfPTbuildcellstyle` command works like `\pgfPTbuildcell`, but stores the cell style under the `name` provided. The constructed cell style is only used if the `name` is passed to the `cell style` option of `\pgfPT`. Otherwise it remains unavailable, unlike the `\pgfPTbuildcell` command which immediately affects the cells of the Periodic Table. There are also some built-in styles such as `pgfPT2lang` which can be used to get the element name in two languages and `pgfPTR` which can be used to replace the atomic weight with the atomic radius.

If needed, the `\pgfPTresetcell` command is used to reset the unit cell to its defaults.

4 Color schemes library

The Color Schemes Library is at present the only library available. This library extends the features provided by the command `\pgfPTnewColorScheme`, which can be used to set the cell background color of each of the 118 elements. The library defines a set of commands that automatically generate a new color scheme:

`\pgfPTGroupColors`

`\pgfPTPeriodColors`

`\pgfPTCScombine`

`\pgfPTCSwrite`

The first two commands, `\pgfPTGroupColors` and `\pgfPTPeriodColors`, act similarly on groups and periods, respectively, and have three arguments, given in the following order:

1. `[default group color]`
   Optional argument to specify the default color for groups if it is not defined in `colorlist`.
2. `{name of the new color scheme}`
   First mandatory argument with the name of the color scheme.
3. `{list of colors,options}`
   Second mandatory argument with the set of colors for the groups or periods and, if desired, some options acting on the set of colors.

   The color set can be defined by:

   - setting the colors one by one using the syntax `G1=`⟨*color*⟩ , ... , `G18=`⟨*color*⟩ for groups or `P1=`⟨*color*⟩ , ... , `P7=`⟨*color*⟩ for periods.
   - setting a gradient using the syntax `left color=`⟨*color*⟩, `middle color=`⟨*color*⟩ and `right color=`⟨*color*⟩ where at least one of the color assignments is required. For the `\pgfPTPeriodColors` command use `top` and `bottom` instead of `left` and `right`.
   - setting a custom gradient as the color list by using the `gradient={`⟨*colors*⟩`}` syntax, where color assignments are done with the ⟨*group (or period) number*⟩=⟨*color*⟩ mechanism.

   The options available are:

`H=`⟨*color*⟩ to set the color of the hydrogen cell.

`La=`⟨*color*⟩ to set the color of the lanthanum cell.

`Lanta=`⟨*color*⟩ to set the color of the lanthanoids' cells.

`Ac=`⟨*color*⟩ to set the color of the actinium cell.

`Actin=`⟨*color*⟩ to set the color of the actinoids' cells.

`period|group blending` to perform a color blend over the periods / groups using the commands `\pgfPTGroupColors` / `\pgfPTPeriodColors`.

The `blending` option itself may have options:

- `color=`⟨*color*⟩,
- `percentage=`⟨*signed integer*⟩, and/or
- `mode=add|sub|linear`.

For ⟨*color*⟩ specifications, the base package syntax supports

- any `namedColor` and
- `namedA!##!namedB!##!namedC`, where the `##` are percentages to use of the preceding color.

In addition, any color model supported by the `xcolor` package can be specified using a special syntax: `*[model:values]`. For example, `*[HTML:5FA287]` and `*[rgb:.5;.2;.3]`. Note that here color components are separated by semicolons instead of commas, where applicable, as in the `xcolor` package.

The `\pgfPTCScombine` command combines two known color schemes and merges the result into a new color scheme. It has three mandatory arguments, the two known names and the name of the resulting color scheme; and one optional argument, with parameters `prop1:prop2,mode`, to control how the two color schemes are combined and can be used individually or together. The `prop`s refer to the proportional amount of the first and second color schemes used in the combination and the `mode` refers to how the colors are mixed, `add` for additive mixing, `sub` for subtractive mixing and `linear` for linear mixing (as in the `xcolor` package).

```
\pgfPTGroupColors{myGr}{right color=green}
\pgfPTCScombine[3:1]{pgfPTSoft,myGr,mix}
\pgfPT[back color scheme=mix,Z list=all]
```

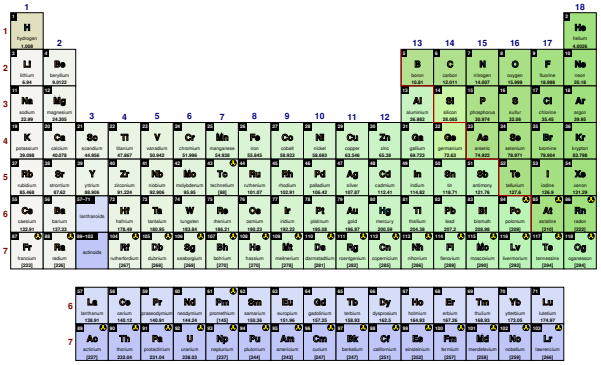


Figure 20: A Periodic Table with the built-in Soft color scheme combined with a user-defined groups color scheme.

Finally the fourth command, `\pgfPTCSwrite`, writes the provided color schemes to a file for later use without loading this library. It has a mandatory argument, the list of the color schemes names to be written and an optional argument, the filename. If no filename is provided the first name in the list of the color schemes names is used. For example, `\pgfPTCSwrite{mix}` will create (or overwrite), in the current working directory, a file named `mix.tex` with the following contents:

```
\pgfPTnewColorScheme{mix}
    {0.88225/0.87924/0.74425,0.63225/...}
```

After that, it can be loaded using `\input{mix.tex}` anywhere in any document (in the same working directory) and the named color scheme will be available for use as usual.

5 Motivation

It all started in 2016: I wanted to improve the Periodic Tables presented in the documents made available to my students. Until then, they were built with drawing software, exported as non-vector images and included in documentation using LaTeX. So, I got down to work and started a local version of a Periodic Table using Ti*k*Z: the unit cell was static and had the atomic number, the chemical symbol, the name, the atomic weight and the radioactivity for the elements up to $Z = 111$. It also had the corresponding legend and the possibility of showing blocks, families and periodic trends. All content was in Portuguese, my native language. For what I intended, I was satisfied.

A couple of years later, with the announcement of the IYPT2019 (The International Year of the Periodic Table), I asked myself: *why not make my TikZ Periodic Table available as a LaTeX package?* Then I started to think... *Well, something in Portuguese will be useful, but it will be more useful if it is also in English. And in French, German, Italian or any other language, as long as I can easily get these translations.* Likewise, I thought the unit cell should be more flexible, allowing other content to be included and where it should be placed. Keeping that in mind I started the job. First I collected the names of elements in five languages, English, French, German, Spanish and Italian, as well as Brazilian Portuguese. After that I had to build a way to use them in the Periodic Table. But the job was only beginning. The next step was to decide what contents would be available. As soon as I made the decision, a new journey began: the construction of the unit cell.

In the summer of 2020 I had a working implementation of the above ideas. That was the first draft of what would become the `pgf-PeriodicTable` package. The cell building mechanism, some of the

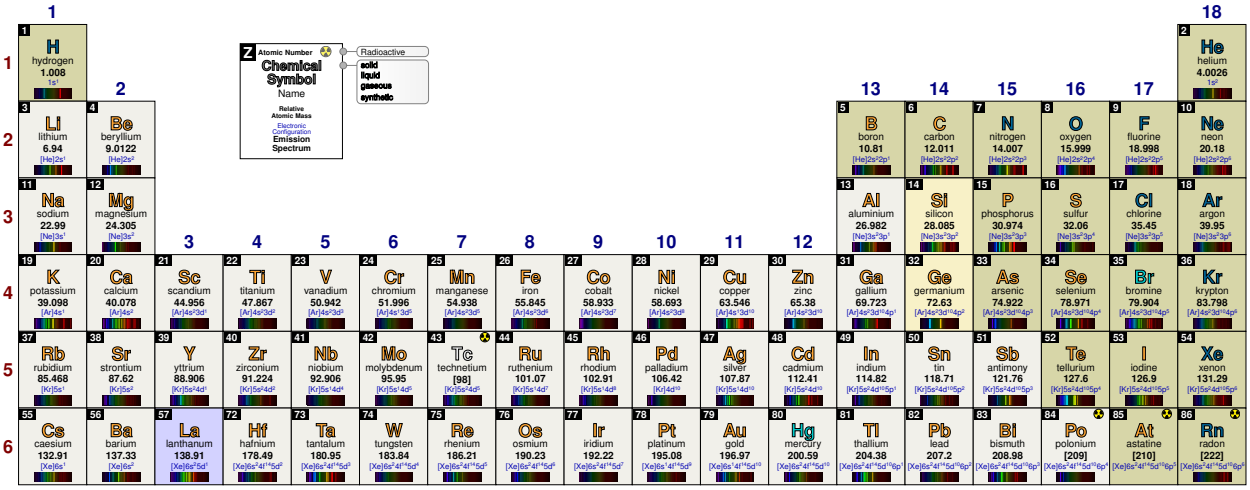


Figure 21: A full-width custom Periodic Table (through radon).

`pgf keys` to customize the Periodic Table and a set of basic data (name, chemical symbol, atomic weight, standard relative atomic mass, radioactivity, atomic radius, first ionization energy, electronegativity and electroaffinity) were implemented.

The next main task was to consider the language support. I had started by collecting the names for the 118 elements in the languages mentioned above. From the collected data, the necessary macros were created to make possible the selection of a language at package loading. The chosen language would be unchangeable after that. But I wasn't yet *convinced*... I had both Portuguese and Brazilian students and I wanted to show them the differences in element names — not only for the Brazilian students, also for the Portuguese ones, because when they searched the Internet for a chemical element's data, for example, most of the results obtained were (and are) in Brazilian Portuguese. So I implemented a trick: instead of different macros for each element in each language, I put the language data in the same macro for each element. This made it possible to dynamically choose the language, not just when loading the package. And, with this mechanism available, the possibility of using simultaneously more than one language was also achieved. So I can build a Periodic Table with element names in Portuguese, both from Portugal and from Brazil, to point out their differences.

The language job finished, the data job began... from November 2021 to July 2022 I added the reviewed data that I had and added more data for all the 118 elements.

The first public release of `pgf-PeriodicTable` package arrived in October 2022: it was sent to and published on CTAN. After that, some bugs were reported and corrected, as well as good suggestions and improvements, that I tried to implement. One of them was from a Dutch user, who asked me if it would be possible to add another language, volunteering to do the necessary translations. Adding new languages was something I already had in mind and this request led me to speed up its implementation. At the moment, those who want to contribute a translation or submit requests or bug reports, can do so at `github.com/HugoPGomes/pgf-periodictable`.

Since the beginning many ideas were growing in my mind for the Periodic Table; some of them have been implemented, others I would still like to implement in the future... For now, let's end with a full-size custom periodic table in figure 21.

References

[1] T. Tantau, C. Feuersänger, et al. *The PGF package*. Create PostScript and PDF graphics in TEX. `ctan.org/pkg/pgf`

[2] Dr. Uwe Kern, The LATEX Project Team. *The xcolor package*. Driver-independent color extensions for LATEX and pdfLATEX. `ctan.org/pkg/xcolor`

[3] W. Robertson, LATEX Project Team. *The Fontspec package*. Advanced font selection in XELATEX and LuaLATEX. `ctan.org/pkg/fontspec`

◇ Hugo Gomes
Lisboa, Portugal
`hugo.parelho (at) gmail dot com`

Integrating Git information into LaTeX documents: gitinfo-lua

Erik Nijenhuis

Abstract

This article presents the gitinfo-lua package, which integrates Git information into LaTeX documents. This package provides macros for document versioning, listing authors, and displaying a change log, enhancing collaboration and traceability in software documentation.

Keywords

LuaLaTeX, Git

1 Introduction

Back in 2016, I first came into contact with LaTeX while studying Software Engineering at University Windesheim Zwolle. The foremost attractive quality of LaTeX for me back then was the fact that it has textual source. This made the documentation sources a perfect candidate to include in a software project itself. While LaTeX and Git already make a great combination for collaboration and traceability, I still had some wishes for making it the best combination.

Simple things I typically missed were leveraging basic information from Git, like the project version and authors. Another wish was displaying the entire change history of the project in the documentation itself, which can be quite challenging.

These wishes materialized at the end of 2023, when I submitted my first LaTeX package to CTAN, named gitinfo-lua [5]. It requires only git to be installed, and works exclusively with LuaLaTeX for the time being. For other compilers, you can refer to gitinfo2 [3], which requires some configuration of Git hooks, or gitlog [4], which still has some limitations for use in production environments.

2 Document versioning

As mentioned earlier, I wanted to have the software project's version and version for the documentation perfectly aligned. This can be simply achieved by versioning those sources all together in a single repository. Take, for example, the following project structure:

```
⟨project path⟩
├─ .git
├─ doc
│  └─ disaster-detection-alarm.tex
├─ src
│  └─ disaster-detection-alarm.c
└─ Makefile
```

A document version can be set by creating a Git tag, as in:

```
git tag -a v1.0 -m "Release version 1.0"
```

Then `git describe --always --tags` will return `v1.0`, which is the command line executed within the package itself. If the tag is dirty, however, it will append the commit count and abbreviated hash to the version. For example: `v1.0-24-gcc4live`, which means 24 commits ahead of the tag and pointing at revision[1] `cc4live` (without the `g`). This way, you'll always be able to reproduce an exact copy of it using Git checkout. For example:

```
git checkout -b recovery cc4live
```

will check out a branch named `recovery` pointing to revision `cc4live`.

So to display the version in the LaTeX document, we'll use the `\gitversion` macro.

```
\documentclass{article}
\usepackage{gitinfo-lua}
\begin{document}
    This is version \gitversion\ of
    project \texttt{disaster-detection-alarm}.
\end{document}
```

which will result in something like:

> This is version v1.0-24-gcc4live of project `disaster-detection-alarm`.

It's important to know that, when making use of `latexmk`'s [1] `--pvc` option, committing changes will cause a rerun, since file `.git/HEAD` is being recorded. This doesn't count for creating tags, so be sure to clean and recompile the entire document after creating a tag.

3 Referencing authors

The package provides two separate implementations for inserting author information.

The first is the macro `\gitauthor` together with `\gitemail`, which comes with a variety of problems. This macro uses the global Git configuration of the local machine. This is problematic on a CI/CD level, because the document author ends up being something like *gitbot* instead of your locally configured name.

For most users, the macros `\dogitauthors` and `\forgitauthors` would be best to use. Both work fine with single authors and leverage the entire Git history to list all authors of the project. The simplest

[1] [Revision is a] synonym for "commit" (the noun). git-scm.com/docs/gitglossary

doi.org/10.47397/tb/45-3/tb141nijenhuis-git

to use is `\dogitauthors`, which optionally takes a conjunction text. For example:

```
\dogitauthors[,~]
```

which could result in something like:
Alice ⟨alice@example.org⟩, Bob ⟨bob@dev.null⟩.

The other macro, `\forgitauthors`, takes a control sequence which gets two arguments: the author's name and the author's email. For example, if for some reason, your typewriter font doesn't support `\textlangle` and `\textrangle`, you might want to replace them with angle brackets instead, like so:

```
\newcommand\myauthorformat[2]{%
    #1
    \ifcsname href\endcsname
        \href{mailto:#2}{<#2>}%
    \else
        \texttt{<#2>}%
    \fi
}
\forgitauthors[,~]{myauthorformat}
```

Then the result would look like:
Alice <alice@example.org>, Bob <bob@dev.null>.

Lastly, when competition is of importance, one might use the package option `contrib`. This will sort authors based on their commits made. However, I personally wouldn't be fooled by commit or 'lines changed' statistics — *less is more.*

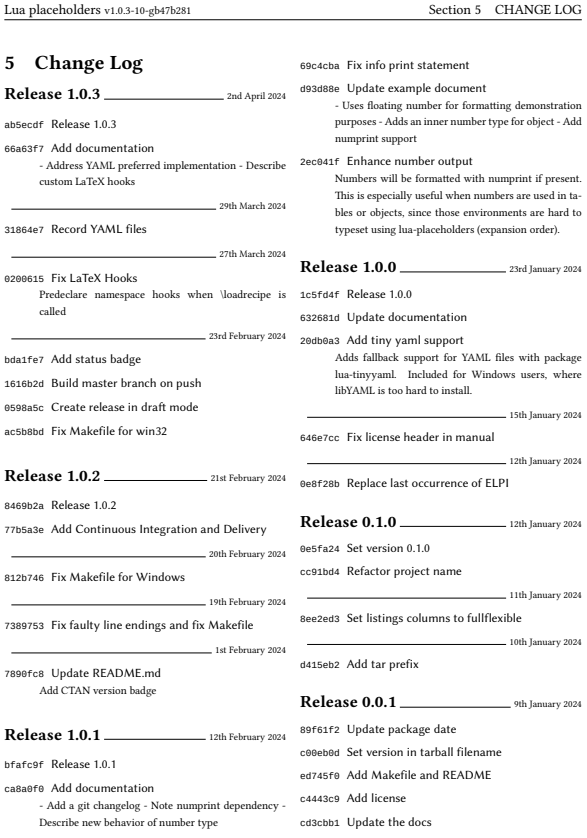
4 Displaying the change history

The most complex feature `gitinfo-lua` provides is formatting the change history in a LaTeX document. For this to work, I had to come up with two iterating commands, namely `\forgittagseq` for iterating over the tags and `\forgitcommit` to iterate over commits between a revision specification, like `v1.0..v2.0`.

To demonstrate this feature, I'll take a real-world example, namely my `lua-placeholders` manual:

```
\newcommand\XerdiLPL
    {https://github.com/Xerdi/lua-placeholders}
\newcommand\setcurdate[1]{\def\curdate{#1}}%
\newcommand\commitline[4]{%
    \ifthenelse{\equal{#2}{\curdate}}{}{%
        \item[] \hrulefill~{\scriptsize
              \printdate{#2}}\def\curdate{#2}}%
    \item[\href{\XerdiLPL /commit/#4}{#4}]%
    {\small\fontfamily{\sfdefault}%
     \selectfont#1}%
    \ifx&#3&\else\\% Commit body
        {\footnotesize#3}%
    \fi
}
\newcommand\formatversion[3]{%
    {\bfseries\large Release
     \href{\XerdiLPL /releases/tag/#1}{#1}}%
    ~\hrulefill~%
    \gittag[(taggerdate)
          (taggerdate:short)(authordate:short)]
         {setcurdate}{#1}%
    {\scriptsize\gittag[(taggerdate)
          (taggerdate:short)(authordate:short)]
         {printdate}{#1}}
    \begin{description}
      [font=\small\ttfamily,itemsep=1pt]
        \forgitcommit[s,as,b,h]{commitline}
                     {#3,flags={no-merges}}
    \end{description}
    \bigskip
}%
\forgittagseq{formatversion}
```

And the output (truncated):



Lua placeholders v1.0.3-10-gb47b281 Section 5 CHANGE LOG

5 Change Log

Release 1.0.3 — 2nd April 2024

ab5ecdf Release 1.0.3

66a63f7 Add documentation
- Address YAML preferred implementation - Describe custom LaTeX hooks

29th March 2024

31864e7 Record YAML files

27th March 2024

0200615 Fix LaTeX Hooks
Predeclare namespace hooks when \loadrecipe is called

23rd February 2024

bda1fe7 Add status badge

1616b2d Build master branch on push

0598a5c Create release in draft mode

ac5b8bd Fix Makefile for win32

Release 1.0.2 — 21st February 2024

8469b2a Release 1.0.2

77b5a3e Add Continuous Integration and Delivery

20th February 2024

812b746 Fix Makefile for Windows

19th February 2024

7389753 Fix faulty line endings and fix Makefile

1st February 2024

7890fc8 Update README.md
Add CTAN version badge

Release 1.0.1 — 12th February 2024

bfafc9f Release 1.0.1

ca8a0f0 Add documentation
- Add a git changelog - Note numprint dependency - Describe new behavior of number type

69c4cba Fix info print statement

d93d88e Update example document
- Uses floating number for formatting demonstration purposes - Adds an inner number type for object - Add numprint support

2ec041f Enhance number output
Numbers will be formatted with numprint if present. This is especially useful when numbers are used in tables or objects, since those environments are hard to typeset using lua-placeholders (expansion order).

Release 1.0.0 — 23rd January 2024

1c5fd4f Release 1.0.0

632681d Update documentation

20db0a3 Add tiny yaml support
Adds fallback support for YAML files with package lua-tinyyaml. Included for Windows users, where libYAML is too hard to install.

15th January 2024

646e7cc Fix license header in manual

12th January 2024

0e8f28b Replace last occurrence of ELPI

Release 0.1.0 — 12th January 2024

0e5fa24 Set version 0.1.0

cc91bd4 Refactor project name

11th January 2024

8ee2ed3 Set listings columns to fullflexible

10th January 2024

d415eb2 Add tar prefix

Release 0.0.1 — 9th January 2024

89f61f2 Update package date

c00eb0d Set version in tarball filename

ed745f0 Add Makefile and README

c4443c9 Add license

cd3cbb1 Update the docs

This example shows release lines and commit lines separated by date. The release lines begin with a hyperlinked title containing the version and end with the release date separated by a horizontal rule. Before the release date is printed, `\curdate` is also updated, which we'll use while grouping the commit lines. After printing the release line, we'll put all commits into a `description` environment.

For every commit, an optional date line is displayed, if the dates differ. A commit line begins with a hyperlinked abbreviated commit hash, followed by the commit message. If present, the commit body is displayed in a smaller font.

So, how does it work? It all begins at the end of the example, where the first macro, `\forgittagseq`, takes a control sequence which gets three arguments, like `\formatversion`. The first argument gives the ⟨*current*⟩ tag, the second the ⟨*next*⟩ tag, and the third a ⟨*rev spec*⟩, which most of the time is the current and next concatenated by two dots, i.e., '`1.0..1.1`'.[2] The last argument is then used for the second macro, `\forgitcommit`, which is called for every tag sequence.

A significant difference between the two macros is that `\forgitcommit` can take extra key-value arguments within its mandatory argument, as the example demonstrates with `flags={no-merges}`. This option filters out all merge commits.

Another meaningful option next to `flags` is `files={...}`, which can be used to filter commits by the given filenames. That can be useful if you only want to include changes in the documentation.

The optional argument for `\forgitcommit` takes a comma-separated list of Git format placeholders.[3] However, as explained in the Git documentation, every placeholder is preceded by a %, which would be cumbersome in LaTeX. To control the arguments you'll get in your formatting macro, you can pass every desired placeholder without the % character separated by commas to the optional argument of `\forgitcommit`. The arguments will be passed to the formatting macro (`\commitline`) in the specified order.

5 Conclusion

As demonstrated, leveraging Git data in LaTeX can be both practical and powerful, providing a transparent representation of the changes made and ensuring the integrity of the documentation. It offers a fully traceable change log that can be perfectly hyperlinked to web solutions like GitHub and can be formatted in any desired structure.

6 Discussion

6.1 Commit message tenses and clarity

When integrating Git information into LaTeX documents, maintaining clear and meaningful commit messages is crucial. Typically, Git commit messages are written in the imperative mood, like giving instructions (e.g., "Fix typo"). This style aligns with Git conventions.

For change logs in documentation, converting these messages to the past tense (e.g., "Fixed a typo in the disaster detection algorithm documentation") can make the history more readable and provide a clear record of what has been done.

Both forms have their merits: imperative mood for consistency with Git and past tense for readability in change logs. Ultimately, it's up to you to decide which form to use, but maintaining consistency within a project is key for clear communication.

6.2 Enhancing macros with `luakeys`

The `gitinfo-lua` package benefits greatly from using `luakeys` [2] for defining complex macro arguments, as `\forgitcommit` does. This provides approximately the same features as `pgfkeys` provides, but within the context of Lua instead of LaTeX 2ε.

6.3 Using Git commands without `--shell-escape`

A common concern when using external commands in LaTeX is the need for the `--shell-escape` option, which can pose security risks. To mitigate this, add the `git` command to the `shell_escape_commands` setting in your `texmf.cnf` file. This allows the necessary Git commands to be executed without enabling the full `--shell-escape` option, enhancing security while maintaining functionality (although nefarious documents could then use `git` to do more or less anything, so be careful).

References

[1] J. Collins, E. McLean, D.J. Musliner. *The latexmk package.* `www.cantab.net/users/johncollins/latexmk/index.html`

[2] J. Friedrich. *The luakeys package.* `ctan.org/pkg/luakeys`

[3] B. Longborough, The LaTeX Project Team. *The gitinfo2 package.* `ctan.org/pkg/gitinfo2`

[4] B. Longborough, The LaTeX Project Team. *The gitlog package.* `ctan.org/pkg/gitlog`

[5] E. Nijenhuis. *The gitinfo-lua package.* `ctan.org/pkg/gitinfo-lua`

◇ Erik Nijenhuis
Frans Halsstraat 38
Leeuwarden, 8932 JC
The Netherlands
`erik (at) xerdi (dot) com`
`https://github.com/MacLotsen`

[2] The last iteration has the current tag as third argument, so that it will show all commits from the tag to the initial commit.

[3] See 'Pretty formats' with `man git-log` to see all possible formatting placeholders.

A color concept for HiTEX

Martin Ruckert

Abstract

While using colors is expensive in print, it is available at no cost on screen. Therefore HiTEX and the HINT file format need support for colors. The design, its objectives, and specification is described in the article. The result is illustrated by examples.

1 Designing a programming API

The current design of color primitives in the various TEX engines is strongly influenced by the available primitives of pdfTEX. The design space available when pdfTEX came into being was limited by the features that the target PDF output format could provide, which in turn was limited by the capabilities of the graphics hardware at that time. The graphics hardware available in the 1990s on personal computers is no match to the capabilities we have now. Even the capabilities of expensive workstations or the dedicated hardware of professional digital printers at that time fall short of what we have today on a mobile phone. The limitations given by the output format were not negotiable. PDF started as a proprietary format of Adobe and then evolved into an international standard. A comparatively small user community like ours had little to no influence on the decisions made.

The necessity to design an new color API for HiTEX offers me a unique opportunity: To design an API with hardly any restrictions. It starts with the available graphics hardware which is dimensioned to handle gaming workloads that far exceed anything that we might need for displaying TEX output. Further, the graphic cards are programmable for example with OpenGL. This puts the raw capabilities of the hardware at the disposal of the programmer without the limitations of a predefined API like, for instance, the Microsoft Windows graphics device interface (GDI). Since I am the designer and implementer of the viewer applications for HINT files as well as the HINT file format, I can freely choose what features the file format should offer. As the designer and implementer of HiTEX, I can start with designing its programming API and work my way back from HiTEX's requirements to the graphics hardware. To be honest, I had to do this journey three times before arriving at the present design. Each time the struggle for an efficient and elegant solution taught me new insights that convinced me to start over again.

2 Design objectives

Of course the new API should be powerful and flexible, elegant and easy to use, but at the same time allow for efficient implementations. Efficiency means that colors should not take up too much space in the HINT file, and rendering the HINT file should not require expensive operations. The efficiency of rendering a HINT file is far more important than the efficiency of creating a HINT file because the former might run on a small mobile, battery powered, device with severe restrictions on memory and energy consumption, while the latter usually runs on a personal computer with lots of memory and processing power of a different order of magnitude.

There were two more complex objectives: The new design should respect the structure and spirit of TEX, and it should be largely compatible with the existing designs of TEX color primitives. The latter is crucial for user acceptance; no one will rewrite or adapt an existing macro package if the benefits do not warrant the effort. The former is crucial for an intuitive use of the primitives: few people read manuals, most people follow their instincts when they modify and extend pre-existing documents. So what the primitives do should be in line with what users generally expect of TEX.

The most important design decision that followed from these objectives was to associate background colors with boxes, not rules. Rules are colored with the foreground color so that for example the fraction line naturally gets the same color as numerator and denominator. Boxes can be stored in registers and should take their background color with them when taken out of the register. Boxes can be nested, and therefore transparent colors must reflect the nesting of boxes.

3 Use of colors in text

Three different decision makers might determine the color of any word or glyph: the document author, the reader of a document, and the rendering application. One important design goal of the HINT file format was to give the document author ultimate control over the appearance of the document. This is in contrast to the HTML file format, where the rendering application, the browser, can decide the fonts or colors, along with many other aspects of the document. The browser in turn offers the user various settings to adapt the appearance to personal preferences. This leads to a permanent fight between document designers and document consumers over how a document should look.

The reader of a document has legitimate requirements for the look and feel of a document. It is

doi.org/10.47397/tb/45-3/tb141ruckert-hicolor

common for user interfaces to offer a "Dark" mode that changes the colors to reflect a dark reading environment. Other users might need a "High Contrast" mode or a "Color Blind" mode. So the HINT file format is capable of offering various modes. Currently there are only two modes implemented, "Dark" mode and "Normal" or "Day" mode; there might be more modes in the future. The user interface of the rendering application should enable choosing between the different modes available to the user.

Also the rendering application might have the need to make a color change. For example if the reader uses a search form, the application might want to highlight all matching words on the page. Just using a yellow foreground color might work sometimes but not if the background happens to be yellow as well. If the background is green, the highlighting might still be visible, but hard to read, and incredibly ugly. And what might work in day mode might no longer work in dark mode. Besides "highlighting", the renderer also has the option to "focus" a part of the document. There might be more choices for the renderer in future versions.

To give the document author full control over the colors used, while still allowing the user and renderer to initiate color changes, a color change in a document, as given by the document author, must specify colors for all possible modes, and for each mode, colors for normal, highlighted, and focus text must be given. Together this ensures that a document might use colors in a decent way, that works well with the user's choices and the renderer's necessities. Of course it is essential that useful default colors ease the burden of specifying all these colors. While the design presented here offers some support for default values, the main responsibility to compute and provide useful defaults lies with high-level macro packages.

4 The specification

This section tries to give a complete specification of syntax and semantics of the primitives envisioned for the color support of HiTEX. To describe the syntax, I will use the extended Backus-Naur form (EBNF). For those not familiar with this formalism, here are a few explanations: A syntactic element is represented by a "symbol". I use ⟨ *italics* ⟩ enclosed in pointed brackets to denote symbols, and I use rules to define the meaning of symbols. A rule starts with the symbol to be defined, followed by a colon "`:`", and then the text that this symbol stands for. A rule ends with a period "`.`". Some symbols refer to text that is defined as part of standard TEX. These symbols are explained by an informal description. For example:

⟨ *integer* ⟩`:` an integer as in `\penalty`⟨ *integer* ⟩`.`
⟨ *number* ⟩`:` a number as in `\kern`⟨ *number* ⟩`pt.`

Optional parts of the rule's text are enclosed in [square brackets]. Alternatives are separated by a vertical bar "|". For text that must occur verbatim in the TEX source file, I use a `typewriter font`.

4.1 Single colors

The internal representation of a color in HiTEX as well as in the HINT file format uses three bytes to represent a color using the sRGB IEC61966-2.1 color space specification and a fourth byte for an alpha channel, which specifies the level of transparency. The sRGB color space is neither the most modern nor the one preferred by professional artists, but it is available (at least approximately) on all computer monitors and these are the main target for displaying HINT files. There are alternative external formats possible, for example using high precision floating point numbers or alternative color space definitions, but at the end, every color needs to be rounded to the best possible internal representation just described.

Now let's consider the external representation of a color when using the HiTEX color primitives.

The most common color specification is the specification of a foreground color. Because we use one byte for each of the four values that define a color, it is common to specify the three color components, red, green, blue, and the transparency component alpha (in this order) using integer values in the range 0 to 255. Using this representation, a foreground color can be specified using the following syntax:

⟨ *foreground* ⟩`:` `FG {` ⟨ *integer* ⟩ ⟨ *integer* ⟩ ⟨ *integer* ⟩ [⟨ *integer* ⟩] `}.`

For convenience, the alpha value is optional; if no alpha value is given, the value 255 will be used and the color is completely opaque.

Here are some examples: `FG{255 0 0}` and `FG{255 0 0 255}` both specify the same plain and opaque red; `FG{0 0 255}` is plain blue; `FG{255 255 0 127}` is a transparent yellow. Because each value fits in a single byte, the values are often given in hexadecimal notation. In TEX, hexadecimal values are written with a `"` prefix. The same colors as before are then written `FG{"FF 0 0}`, `FG{"FF 0 0 "FF}`, `FG{0 0 "FF}` and `FG{"FF "FF "7F}`. Values greater than 255 or less than 0 are not allowed.

A common alternative to the color representation just described is a device-independent notation, where each value is a real number in the interval from 0 to 1. To keep both representations unambiguous, the device-independent representation (with

the smaller numbers) uses the lowercase keyword `fg` instead of `FG`. Here is the syntax:

⟨ *foreground* ⟩: `fg {` ⟨ *number* ⟩ ⟨ *number* ⟩ ⟨ *number* ⟩ [⟨ *number* ⟩] `}`.

Using the new syntax, the colors above are written `fg{1 0 0}`, `fg{1 0 0 1}`, `fg{0 0 1}` and `fg{1 1 0 0.5}`. Values greater than 1 or less than 0 are not allowed.

The big difference between `fg` and `FG` is the possible range of values. This can be especially confusing when using the value 1, which belongs to both value ranges; in the first case 1 is the maximum value, and in the second, the smallest above zero. So `fg{1 1 1}` is pure white, while `FG{1 1 1}` is the darkest possible gray, which on most devices is indistinguishable from pure black.

4.2 Color pairs, sets, and specifications

In the HINT file format colors always are given as a pair: foreground and background.

⟨ *color* ⟩: ⟨ *foreground* ⟩ [⟨ *background* ⟩].
⟨ *background* ⟩: `BG {` ⟨ *integer* ⟩ ⟨ *integer* ⟩ ⟨ *integer* ⟩ [⟨ *integer* ⟩] `}`.
⟨ *background* ⟩: `bg {` ⟨ *number* ⟩ ⟨ *number* ⟩ ⟨ *number* ⟩ [⟨ *number* ⟩] `}`.

The rules for the background color mirror the rules for the foreground. Note that the specification of the background is optional. Here and in the following, default values are used if optional values are not given.

Here are some examples: `fg{0 0 0}` specifies the foreground color black which is then used for rules and glyphs. In addition to the foreground color, you can specify a background color. For example, black text on white background is specified by `fg{0 0 0} bg{1 1 1}` or, using hexadecimal, `fg{0 0 0} BG{"FF "FF "FF}`.

Colors always come as a color set where a color set consists of three color pairs: for normal text, for highlighted text, and for text that has the focus.

⟨ *color set* ⟩: ⟨ *color* ⟩ [⟨ *color* ⟩ [⟨ *color* ⟩]].

The second and third color pair are again optional.

Finally, a call to the `\HINTcolor` primitive that is used in HiTEX to change the current color has the format: `\HINTcolor {` ⟨ *color specification* ⟩ `}`. A ⟨ *color specification* ⟩ will comprise colors for all color modes available to the reader.

⟨ *color specification* ⟩:
⟨ *color set* ⟩ [`dark` ⟨ *color set* ⟩].

For example, if in dark mode you like white letters on a deep blue background you can write `\HINTcolor {fg{0 0 0} bg{1 1 1} dark fg{1 1 1} bg{0 0 0.3}}`. Such a color specification needs $4 * 2 * 3 * 2 = 48$ bytes, and with more modes, even more bytes. To reduce the memory use for colors, color specifications are listed in the definition part of a HINT file and referenced in the content part of the HINT file by a one byte number. This limits the number of different color specifications (not colors) in a single document to 256.

4.3 Colors and boxes

The renderer of a HINT file will render boxes, glyphs, and rules from left to right and top to bottom (right-to-left languages are not currently supported in HINT). The colors given with the `\HINTcolor` primitive will have an immediate effect on all boxes, rules, and glyphs that follow in the direction of rendering. At any point inside a box there is exactly one current foreground color and one current background color.

Rules and glyphs are rendered with the foreground color on top of the background color. The effect of a color change will persist until the next change of colors or until the end of the box — whichever occurs first. Restricting the effect of a color change to the enclosing box implies a natural nesting of colors that follows the nesting of boxes: An inner box is rendered on top of the outer box.

The implementation of the renderer must follow this rendering order. While the rendering of glyphs and rules in the current foreground color is simple, rendering the background requires extra effort because the background must be rendered before rendering glyphs, rules, and inner boxes on top of it. A new background color will start where the color change is found and fills a horizontal box from top to bottom and a vertical box from left to right.

To render the background, the renderer needs to know where the new background color should end. There are two choices: searching the document for the next explicit or implicit color change, or buffering all rendering activities until the end of the current background is reached. The first alternative is easy to implement and since the rendering is always concerned with a single page the search is quite fast. The second alternative is more complex to implement, but possibly much more efficient because transferring a whole array of rendering data from main memory to the graphic card tends to be much faster than writing each item separately.

4.4 HiTEX's color stack

While the HINT file format can represent nested boxes, which affects the rendering of colors as just described, the color model of the HINT file system within a box

is completely flat. But TEX is a document description where nearly everything can be nested inside everything else. So having a color stack that allows the author or macro programmer to say: "return to the previous color" seems indispensable. So HiTEX maintains a color stack and a few other features that allow for more convenient programming.

The `\HINTendcolor` primitive of HiTEX relies on this color stack. It will look up the color specification that was valid just before the matching use of `\HINTcolor` and insert in the HINT file a color change that returns to this previous color. If there is no matching `\HINTcolor` primitive, the `\HINTendcolor` primitive is silently ignored. Note that within a single box, there is at any point only a single background color: The color stack will switch from one background color to another background color but will not overlay an "inner" background color over an "outer" background color. This is the case only when multiple boxes are nested as described above.

Further, splitting off the initial part of a vertical box with `\vsplit` will insert a color node in the remaining part if necessary to keep the color consistent across the split.

Complications arise from color changes in the top-level vertical list, which is split into pages in the HINT file viewer at runtime. Because the page builder in the viewer has no global information and should not need global information, HiTEX will insert copies of the local color information after every possible breakpoint in the top level vertical list. This will ensure that page breaks will not affect the colors of the displayed material.

However, since TEX considers glue (and kerns) as discardable, it removes such items from the top of a new page. Because glue and kern items are colored using the current background color, they might be visible on a page but disappear when they follow immediately after a page break. So, if you want the effect of a colored glue or kern that is not affected by a page break, you should include it inside a box or use a colored rule instead.

The line breaking algorithm that is part of the HINT file renderer also tracks changes in color within a paragraph and reinserts an appropriate color change at the start of every `\hbox` that contains a new line. In this way local color changes inside a paragraph can span multiple lines but do not affect the interline glue or material that is inserted with `\vadjust`.

4.5 Colors and links

The most common change in color is caused by the use of links. To support this changing of colors, the primitives `\HINTstartlink` and `\HINTendlink` cause an automatic color change. Whenever the `\HINTstartlink` primitive is used, its effect on the colors is equivalent to executing the `\HINTcolor` primitive using the current link color, which is set with the primitive `\HINTlinkcolor {` ⟨ *color specification* ⟩ `}`. This implies that the color change caused by `\HINTstartlink` is local to the enclosing box.

Whenever the `\HINTendlink` primitive is used, it will restore the color stack of HiTEX to its state before the matching `\HINTstartlink`. It is the responsibility of the TEX source code to keep sequences of `\HINTstartlink`, `\HINTendlink`, `\HINTcolor`, and `\HINTendcolor` properly nested. A sequence like `\HINTstartlink` ... `\HINTcolor` ... `\HINTendlink` ... `\HINTendcolor` is not an error, but will cause `\HINTendlink` to restore the colors to those in effect before the `\HINTstartlink`. Then, the following `\HINTendcolor` will either restore the color of a matching `\HINTcolor` preceding the link in the same box or it will restore the color in the outer box, or it will be ignored. In short, color changes inside a link stay local to the link.

4.6 Color defaults

The HINT file format specifies default values for all colors. To override these defaults, HiTEX provides the primitive `\HINTdefaultcolor {` ⟨ *color specification* ⟩ `}`. This primitive must be used before defining any custom colors using `\HINTcolor`.

The HINT format specifies the following default colors: Normal text is black `FG{0 0 0}`, highlight text is slightly dark red `FG{"EE 0 0}`, and focus text is slightly dark green `FG{0 "EE 0}`. The background is transparent white `bg{1 1 1 0}`. In dark mode the background is transparent black `bg{0 0 0 0}`, normal text is white `fg{1 1 1}`, and a lighter red `FG{"FF "11 "11}` and green `FG{"11 "FF "11}` are used for highlighted and focus text.

For convenience, the HINT file format specifies default colors for links as well: in normal mode, links use dark blue `FG{0 0 "EE}` instead of black; in dark mode, links use light blue `FG{"11 "11 "FF}` instead of white. The primitive `\HINTdefaultlinkcolor {` ⟨ *color specification* ⟩ `}` can partly or completely redefine these defaults.

5 Examples

The first example illustrates a common application: Color is used to emphasize a word and accentuate selected paragraphs.

```
\def\redTeX{%
  \HINTcolor{fg{1 0 0}}\TeX\HINTendcolor}
\def\note{\HINTcolor{fg{0.3 0.3 0.3}}}
```

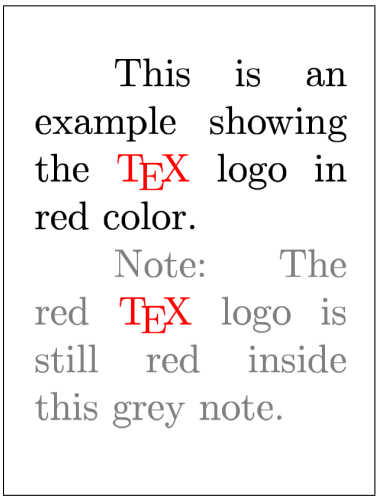


Figure 1: Interacting colors

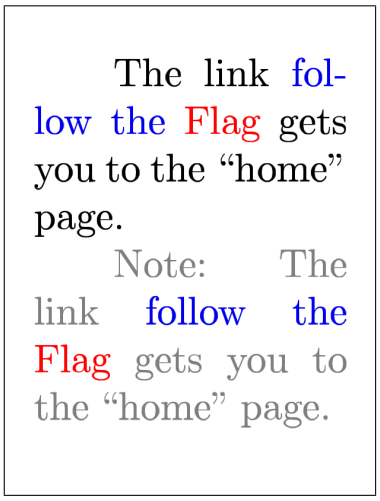


Figure 2: Link colors

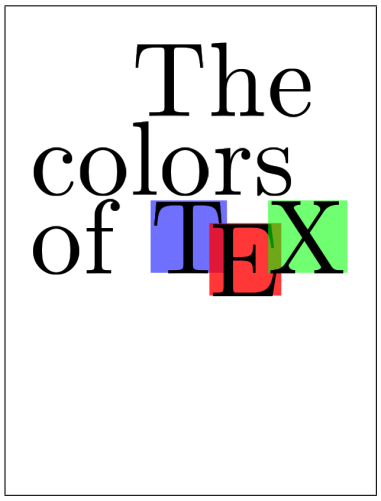


Figure 3: Transparency

```
\def\endnote{\HINTendcolor}

This is an example showing
the \redTeX\ logo in red color.

\note\ Note: The red \redTeX\ logo is still
red inside this grey note.\endnote
```

I show the TEX logo in red and use a `\note` environment that renders text in light gray. The example shows how to implement the color changes, and Figure 1 shows how they interact.

The next example uses `\red` to change the color to red and the “Note” environment from the previous example. It illustrates the interaction with links that are rendered by default in blue.

```
\def\red{\HINTcolor{fg{1 0 0}}}
\def\home#1{%
  \HINTstartlink goto name {HINT.home}
  #1\HINTendlink}

The link \home{follow the \red Flag}
gets you to the ‘‘home’’ page.

\note Note: The link \home{follow the
\red Flag} gets you to the ‘‘home’’ page.
\endnote
```

Figure 2 shows how the end of the link automatically restores the color that was in effect before the link and that hyphenation and line breaking interacts smoothly with color changes.

The last example illustrates the use of transparent backgrounds.

```
\def\blue{\HINTcolor
   {fg {0 0 0} bg{0 0 1 0.3}}}
\def\red{\HINTcolor
   {fg {0 0 0} bg{1 0 0 0.6}}}
\def\green{\HINTcolor
   {fg {0 0 0} bg{0 1 0 0.3}}}
\def\TeX{\hbox{\blue T}\kern-.1667em
  \lower.5ex\hbox{\red E}%
  \kern-.125em\hbox{\green X}}

The colors of \TeX
```

This defines a TEX logo that uses three different background colors for the letters, using alpha values. As seen in Figure 3, the normal kerning and shifting of the letters in the logo causes the backgrounds and the glyphs to overlap producing the effect of overlapping pieces of colored glass.

6 Conclusion

The HiTEX color support is planed to be released with TEX Live 2025 as an experimental release. It will require not only a new version of HiTEX but also an update of the viewer applications for various operating systems (GNU/Linux, Windows, macOS, iOS, Android) so there is still a lot of work to do.

To use the color support with LATEX, driver files need to be written for at least the most important packages that deal with colors. The help of the LATEX team and package maintainers would be greatly appreciated. I assume that these activities will result in new insights that will lead to an improved API and an improved implementation, which might be available as soon as the 2026 TEX Live release.

◇ Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München
Germany
martin.ruckert (at) hm dot edu

Twin demerits

Hans Hagen, Mikael P. Sundqvist

Upgrading math support in ConTEXt not only concerns rendering formulas but also breaking formulas across lines. For instance, fenced formulas should cross lines while retaining the automatic scaling of fences but at the same time you don't want a single fence at the beginning or end of a line. Longer formulas should preferably break somewhere away from the begin and start. Single atoms should not end up at the end of a line and the same is in fact true for text. The later can be prevented by so-called toddler penalties. And then there are languages where (binary) operators need to be repeated, in a similar way as hyphens, at the start of a broken line. Alternative content (swapping one word for another in order to get a visually better looking paragraph) is also possible but those are more (usable) proofs of concept than features used daily. We are fans of the 'rewrite if needed' approach, but it is of course a nice and fun challenge to solve some typographical problems in a generic way, when possible.

So, looking at the end of a broken line and the beginning of the following becomes second nature when moving forward with development. In order to explore and test all these possibilities, we added ways to trace the process of breaking lines: we love to visualize such things. When playing with this we also looked at the start and end of lines with repeated sequences, for instance avoiding the same words or math variable stacking at the start or end, similar to repeated hyphens. But we had enough on our plate to not fully explore this beyond some experiments.

Around that time we had some email contact with Didier Verna, who at the 2023 TUG meeting reported on some experiments he conducted in the ETAP (Experimental typesetting algorithms platform) software that he develops. He followed up on that in 2024 with a preprint of an article reporting on further experiments, especially avoiding similar words at the start and end of successive lines.[1] Of course given the long history of TEX it is no surprise that the wish to avoid that has been expressed before, but this was the first time we have seen detailed data on the topic. We already knew that extensions to the par builder didn't come at a huge performance hit and Didier, also knowing this, therefore wondered if adding such prevention to the engine was an option.

[1] Similarity Problems in Paragraph Justification, An Extension to the Knuth-Plass Algorithm, Didier Verna, EPITA Research Laboratory, Le Kremlin-Bicêtre, France, July 2024 (preprint).

Before we dive into this one should notice that over time many suggestions have been made with regards to where TEX can be improved. Among the reasons why these never made it into the engine are that TEX is frozen, so extensions have to go into successors like ε-TEX, pdfTEX, XETEX, LuaTEX, LuaMetaTEX, etc. One complication is in the way TEX is programmed: it uses a linked list of nodes for what eventually becomes a list of lines, a paragraph. This is a forward-linked list so one cannot look back, although in some cases TEX keeps a pointer to the previous node around. But looking back a 'word' demands quite a bit of extra code. In LuaTEX and LuaMetaTEX we have a dual linked list so there we can go back. This means that implementing a feature as discussed here is less hard and also can be prototyped in Lua. On top of that, in LuaMetaTEX we also carry around more information to act upon. Of course it doesn't change the fact that while experiments can show that 'it can be done' doesn't mean that we don't run into complications that have to be dealt with in order to make it usable and not backfire with bad results elsewhere. We will show a few cases that demonstrate that one reason for engines not to support this out of the box is that for a single (extra) feature like this, one likely has to add far more control options. So keep this in mind when reading on: there is always more involved than at first sight. What looks like a home run for LuaMetaTEX is less so for other engines.

As we had been playing a bit with tracing and analysing decisions we have a mechanism in place to plug code into the par builder. We use this for instance to force breaks based on feedback. Discouraging a break at similar words can be done using those same hooks so we decided to take the challenge and just made it a more permanent feature, possibly with the side effect of it being more integrated in the engine.[2] Below are some outcomes that can be seen as a progress report on this feature.

The first thing we did was go back to the already existing callback. Because the builder is rather complex (keep in mind that we have several extensions) there are nine places where the callback can be triggered, mysteriously identified as: initialize, start, list, stop, collect, line, delete, report, and wrap up. Each call to the same callback gets a different set of status parameters that at that particular moment

[2] We often keep experimental code around, interfaced not at the user level but via runtime directives, if only because we need it for articles. Supporting a feature as discussed here needs some thinking with respect to integrating in, for instance, the paragraph rendering setups which differs from low-level directives.

doi.org/10.47397/tb/45-3/tb141hagen-twins

make sense. It is up to Lua code to collect, analyze, feedback and/or use it somehow. This plugin mechanism seems like a lot of overhead but as it is only needed for tracing it goes unnoticed.

When we play with these repeated words we distinguish between what we call left and right edge twins.[3] We look at glyphs as well as discretionaries and ignore font kerns. We need to check the pre, post and replacement parts of discretionary nodes because we must assume that more complex OpenType features might give a more complex discretionary than a single hyphen.[4]

Using a Lua approach is quite flexible and permits nice tracing but, as said, it abuses a callback that, due to the many different invocations, is not the best candidate for this. We could add another callback but that is overkill. Therefore, after testing, part of the Lua code has been turned into a native feature so that we can do both: native twin checking as well as tracing of the break routine (which we need for testing par passes) but also exploring more variants using that callback.

So, the reference implementation is still done in Lua where we then also have twin tracing. In principle that is fast enough; the overhead on a 240 page (1000) Tufte quote test is around .1 seconds. The native C code implementation works slightly differently but is derived from the Lua code. In the engine we have some constraints, such as limiting the maximum length of a snippet to 16 characters.

In both cases (Lua and C) the overhead is rather small because we look only at a limited set of breakpoints. In Lua we gain performance by caching, in C code by limiting the snippet. We can squeeze out some more performance if needed by immediately comparing the second snippet with the first one. Unlike the Lua variant, the C code implementation checks for a so-called glyph option being set.[5] Because it has to fit into how we handle linebreak controlling parameters, we carry new `\lefttwindemerits` and `\righttwindemerits` registers in the paragraph state node and we can also set it in the (optional extra) paragraph passes, so that it can be disabled when we get bad results. This makes a relatively small patch a bit larger due to housekeeping.

With support in the engine (C code) as well as in Lua (the callback), we can now come back to some of the observations we made when we discussed this feature during experiments. But first let's stress that adding this feature to the engine makes sense so that users can play with it, but this doesn't mean that it always solves the problem. Also, like other features, one might only benefit in a few places in hundreds of pages of text. One should always visually check the result.

In his article Didier uses a quote (from Grimm Brothers' "Frog King") that in his case has three 'and''s in row (using an eight bit Computer Modern font). Actually there are four 'and''s close together that can team up. Here we use a different setup with the same quote. We have different defaults for e.g. tolerance and spacing in ConTEXt anyway. In figure 1, we start with the paragraph as it comes out normally, using 12 point Latin Modern and an `\hsize` of 82mm.[6]

> In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.

Figure 1: Original text.

In figure 2 we show what we get when we set the demerits to 7500 thereby enabling twin detection. This number is pretty high because demerits are usually large numbers, as in squared penalties. When a paragraph is broken into lines TEX keeps track of reasonable breakpoints. As it goes over the paragraph breakpoints get identified and depending on criteria previous breakpoints get looked at. That means that at any of these points we can check if there are similar words before and/or after a pair. If that is the case one or both extra demerits get added to the current accumulated amount. Normally the current amount is in the thousands so that is why we need relatively high twin values.

[3] So in addition to widows, clubs, toddlers and orphans we now have twins too.

[4] In his preprint Didier only mentions glyphs and stops in his experiments at discretionary nodes.

[5] Glyph options control features like kerning, ligature building, protrusion, expansion, at the individual glyph level.

[6] Editor's note: For *TUGboat*, we graphically scaled all examples, including the surrounding frames, to the *TUGboat* column width, which is just over 79mm. The typesetting results are not affected.

In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.

Figure 2: Twin demerits parameters set to 7500, engine implementation.

In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.

Figure 3: Lua implementation, with colored snippets.

In figure 2 we use the engine variant; in the next example we use the Lua implementation, which permits coloring the snippets that we found troublesome. Tracing also happens on the console and that is why we use the callback: if we report the words that matter, we need a proper Unicode string and in a typeset paragraph we might have (in the case of ConTEXt) private ones that point to ligatures, case variants, stylistic alternates etc.

Notice that we not only detect an 'and' case here but also a hyphenated part of 'forest'. Of course the whole 'forest' could also have shown up as a candidate.

All this depends a lot on the fonts and widths used. In figure 4 we use the Pagella font. It demonstrates that one cannot simply assume that when twins get set up the desired effect occurs. Again we set the values to 7500, and in figure 5 we get the same results, contrary to the Latin Modern case.

In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.

Figure 4: The Pagella font, without twin detection.

In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.

Figure 5: The Pagella font, with twin detection, but line breaks are unchanged.

Figure 6 uses the Lua variant so that we can show the candidates, red for the right ones; later we'll also see green for the left ones and yellow for both left and right. In all cases, words are colored when they were considered as twins at some point in the paragraph processing, even if those particular line breaks were discarded later. Thus, the colored words might show up anywhere in a paragraph.

At any rate, the reason why it doesn't work out here is that we need to bump the tolerance and also permit emergency stretch. This shows that just enabling a feature doesn't guarantee results. So, figure 7 does that: more tolerance and possible emergency stretch. Playing with the widths shows that single point differences can have quite some effect.

In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.

Figure 6: Showing candidates in the Pagella example.

In olden times when wishing still helped one, there lived a king whose daughters were all beautiful; and the youngest was so beautiful that the sun itself, which has seen so much, was astonished whenever it shone in her face. Close by the king's castle lay a great dark forest, and under an old lime-tree in the forest was a well, and when the day was very warm, the king's child went out into the forest and sat down by the side of the cool fountain; and when she was bored she took a golden ball, and threw it up on high and caught it; and this ball was her favorite plaything.

Figure 7: With more tolerance and emergency stretch.

more and efficiency and efficient more and efficiency and efficient more and effi-
ciency and efficient more and efficiency and efficient more and efficiency and effi-
cient more and efficiency and efficient more and efficiency and efficient more and
efficiency and efficient more and efficiency and efficient more and efficiency and
efficient more and efficiency and efficient more and efficiency and efficient more
and efficiency and efficient more and efficiency and efficient more and efficiency
and efficient more and efficiency and efficient more and efficiency and efficient
more and efficiency and efficient more and efficiency and efficient more and effi-
ciency and efficient

Figure 8: With demerit registers set to 5000.

Finally we show a few examples with nonsense text. The red words are the ones that show up at the right, the green ones on the left, but when a word can occur at both ends yellow is used.

With the demerits set to 5000 (figure 8) we still get a few 'efficien' at the left but they are different words. One can argue that we could use some snippet length (say six glyphs) but of course then something

more and efficiency and efficient more and efficiency and efficient more
and efficiency and efficient more and efficiency and efficient more and ef-
ficiency and efficient more and efficiency and efficient more and efficiency
and efficient more and efficiency and efficient more and efficiency and effi-
cient more and efficiency and efficient more and efficiency and efficient more
and efficiency and efficient more and efficiency and efficient more and ef-
ficiency and efficient more and efficiency and efficient more and efficiency
and efficient more and efficiency and efficient more and efficiency and ef-
ficient more and efficiency and efficient more and efficiency and efficient

Figure 9: With unusual `\parfillskip` and demerit registers at 25000.

more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient
more and efficiency and efficient

Figure 10: Narrower `\hsize`.

else will bother us. In the next variant of the above, we set `\parfillskip` such that we have a different solution space, combined with an extreme 25000 demerits (figure 9). In both examples we use a 426 point width.

Going narrower, as in figure 10, brings us words that can be at either end (shown in yellow) and leaves us without solution but that is what we expect. One can conclude that this feature works best with a wider layout and is not that useful in columns, unless one prefers excessive space glue over no twins, but the good news is that TeX is unlikely to favor that.

So what can we conclude? First of all that it is indeed possible to get rid of repetition. To what extent this improves a document while not introducing suboptimal paragraphs we leave to the user; Didier makes a case for it. Performancewise,

19 Three bowls made after the fashion of almonds in one branch, a knop and a flower; and three bowls made like almonds in another branch, a knop and a flower: so throughout the six branches going out of the candlestick.

Figure 11: Example from the bible.

8 And thou shalt bring the meat offering that is made of these things unto the LORD: and when it is presented unto the priest, he shall bring it unto the altar.

Figure 12: With perhaps too much font expansion.

5 And every plant of the field before it was in the earth, and every herb of the field before it grew: for the LORD God had not caused it to rain upon the earth, and there was not a man to till the ground.

Figure 13: Overlaying results; the first line ends with a doubled 'the'.

there is no reason not to enable it. We did some tests with the larger documents that we also use for testing other features (math line breaking, page building) and when there are twins seen sometimes they do indeed get separated.

One of our test documents, the King James bible in two columns using the Unifraktur font, is a good candidate but although we find candidates only in some cases the line break routine was not always influenced by the increased demerits. Examples of two letter words are 'of' and 'is' and of course it being English we find plenty 'the' and 'and' but some still ended up below each other, simply because we have narrow columns. In figure 11 (a bitmap screenshot) we see an interesting case but one that happened to render the same without twin detection. Words like 'their' and 'shall' happily team up as twins, no matter how high we set the demerits.

In the pdfTEX project, font expansion was tested on an annotated bible and the combination of text, notes, numbers, references was a real challenge.[7] In the abovementioned King James we don't have those constraints but one can wonder what setup will make the verse in figure 12 come out better. We bet that the double twins here are considered less of a problem than excessive spacing or extreme expansion.

The good news is that in this 740 page document, there were quite a few catches, like the one in figure 13. In some cases we got one line more or less and therefore a different column or page break. Of course this itself then can create a problem, like a widow or club but we set that up with pretty high penalties and combined with vertical expansion and page slack tolerance (both are relatively new features) we still get good output so overall we gain in quality!

We tested two other documents that show some interesting (challenging) aspects. In Mikael's university course we compared the versions with and without twin control. The tracer identified 25 situations where demerits could be bumped. We noticed that 'att', 'och', 'så', 'vi' and other short words were popular candidates but after turning on tracing we saw that many were left and were surprised to see quite a few longer words, with of course in a math book quite some 'komplexa' and 'negativa' showing up at the edge.

[7] Hàn Thế Thành's thesis reported on that.

element (a, b) där $a \in A$ och $b \in B$. Exempelvis är $\mathbb{N} \times \mathbb{N}$ mängden av alla par av naturliga tal, såsom $(1, 1)$, $(2, 3)$ och $(101, 23)$.

Ett mycket viktigt och centralt begrepp i matematiken är begreppet *funktion*. Om A och B är två mängder, så tänker vi ofta på en funktion från A till B som en regel som till varje element i A tilldelar ett entydigt bestämt element i B. Om f är en funktion från A till B, så skriver vi $f\colon A \to B$. Det element i B som funktionen f tilldelar ett element $a \in A$ betecknas $f(a)$. Mängden A kal-

Figure 14: Math example; the 'alla' on the first line is repeated at the bottom of the previous page.

Funktionen $x \mapsto \tan x$, kvoten av $\sin x$ och $\cos x$ är väldefinierad så länge som $\cos x \neq 0$. Det är klart att $\tan 0 = 0$ och att $\tan$ är kontinuerlig och strängt växande på $(0, \pi/2)$ (strängt växande följer av att sin är positiv och strängt växande och cos är positiv och strängt avtagande på intervallet). Då sin är

Figure 15: Worse math example.

Keep in mind that we only look at a subset of the possible breakpoints. Of these 25 only 5 were applied, so for the other 20 the solution space was not adequate. For the 5 cases the solution resulted in somewhat narrower lines so we wondered if additional par passes made (hz) expansion kick in but it didn't so in the end we're okay. Of the 20 remaining cases 10 had long words, some with hyphenation so actually we had more cases. An interesting side effect of tracing (by color) is that we noticed that the long words also had successive words and that rewriting the paragraph made sense.

In a math document sometimes it's unavoidable. In figure 14 we see a few troublemakers and 'alla' is actually not resolved. The figure shows the top of a page and at the bottom of the previous page there's also 'alla'. We don't even want to ponder how to bring page breaks into this model. One can also wonder what is more troublesome: edge cases or middle cases.

Also worth noticing is that when twins end up in the middle they tend to stack even when the par builders in the end didn't consider the end-of-line case anyway. A bad example had three separate slightly offset but still stacked long words, shown in figure 15. And, once the author saw this, he made a note to "fix it by rephrasing".

The 'solved' cases were mostly short words but so were unsolved ones; see figure 16. The constraints that math put on breaking the lines win over any twin constraints we add. We also were confirmed in our decision to take discretionaries into account.

Om $0 < a < 1$ så är $1/a > 1$ och $a^x = 1/(1/a)^x$. Alltså följer det från det vi just gjort att $(1/a)^x \to +\infty$ då $x \to +\infty$, dvs. givet A existerar ω så att $(1/a)^x > A$ om $x > \omega$. Givet ett godtyckligt $\varepsilon > 0$ finns det alltså ω så att $|1/(1/a)^x - 0| = 1/(1/a)^x < \varepsilon$ om $x > \omega$. Alltså gäller det att $a^x = 1/(1/a)^x \to 0$ då $x \to +\infty$. ∎

Figure 16: Math example.

—superficially at least—by a visible wound to his head; such a wound as might have been produced—and as, on the final evidence, had been —by a fatal slip, in the dark and after leaving the public house, on the steepish icy slope, a wrong path altogether, at the bottom of which he lay. The icy slope, the turn mistaken at night and in liquor, accounted for much—practically, in the end and after the inquest and boundless chatter, for everything; but there had been matters in his life—strange passages and perils, secret disorders, vices more than suspected—that would have accounted for a good deal more.

Figure 17: From *The Turn of the Screw*.

settled: there was a queer relief, at all events—I mean for myself in especial—in the renouncement of one pretension. If so much had sprung to the surface, I scarce put it too strongly in saying that what had perhaps sprung highest was the absurdity of our prolonging the fiction that I had anything more to teach him. It sufficiently stuck out that, by tacit little tricks in which even more than myself he carried out the care for my dignity, I had had to appeal to him to let me off straining to meet him on the ground of his true capacity. He had at any rate his freedom now; I was never to touch it again; as I had amply shown, moreover, when, on his joining me in the schoolroom the previous night, I had uttered, on the subject of the interval just concluded, neither challenge nor hint. I had too much, from this moment, my other ideas. Yet when he at last arrived, the difficulty of applying them, the accumulations of my problem, were brought straight home to me by the beautiful little presence on which what had occurred had as yet, for the eye, dropped neither stain nor shadow.

Figure 18: Another text from *The Turn of the Screw*.

We also tested a document that Mikael typeset from Gutenberg sources for a book club, Henry James' *The Turn of the Screw*. Here we again noticed quite a few duplicates but also quite a few eventually separated twins, as in figure 17.

In figure 18 we wondered if the twin handler had kicked in which indeed was the case. But we also noticed that without this mechanism being enabled, the same midline stacking occurred. However, in both cases, without coloring they can easily go unnoticed; just try to locate them in figure 19. (See figure 22 for the results.)

This document also demonstrated that words close together tend to register as siblings, and when Mikael showed one of his children what we were looking at, she noticed disturbing repetitions which we hadn't noticed before.[8] But adding more tricky mechanisms will only make the solution space smaller so we will not reveal every annoyance. We did once

[8] From figure 20 you can deduce what words were involved. In that example there are many possible twins, so we set `\twinslimit` to 3, a feature added for this purpose to the Lua version.

was a queer relief, at all events—I mean for myself in especial—in the renouncement of one pretension. If so much had sprung to the surface, I scarce put it too strongly in saying that what had perhaps sprung highest was the absurdity of our prolonging the fiction that I had anything more to teach him. It sufficiently stuck out that, by tacit little tricks in which even more than myself he carried out the care for my dignity, I had had to appeal to him to let me off straining to meet him on the ground of his true capacity. He had at any rate his freedom now; I was never to touch it again; as I had amply shown, moreover, when, on his joining me in the schoolroom the previous night, I had uttered, on the subject of the interval just concluded, neither challenge nor hint. I had too much, from this moment, my other ideas. Yet when he at last arrived, the difficulty of applying them, the accumulations of my problem, were brought straight home to me by the beautiful little presence on which what had occurred had as yet, for the eye, dropped neither stain nor shadow.

Figure 19: Without colorizing.

saying that instead of growing used to them—and it's a marvel for a governess: I call the sisterhood to witness!—I made constant fresh discoveries. There was one direction, assuredly, in which these discoveries stopped: deep obscurity continued to cover the region of the boy's conduct at school. It had been promptly given me, I have noted, to face that mystery without a pang. Perhaps even it would be nearer the truth to say that—without a word—he himself had cleared it up. He had made the whole charge absurd. My conclusion bloomed there with the real rose flush of his innocence: he was only too fine and fair for the little horrid, unclean school-world, and he had paid a price for it. I reflected acutely that the sense of such differences, such superiorities of quality, always, on the part of the majority—which could include even stupid, sordid headmasters—turn infallibly to the vindictive.

Figure 20: Example of many twins, with `\twinslimit=3`.

consider `\siblingpenalty` but already forgot what for, but we hereby reserve that name.

There is plenty left to explore. It is not uncommon in the TEX community to hear users (and developers) express the wish for a feature, offer a few examples of why it's needed, and then fall silent. Time and money can be arguments used to not spend time on actually implementing something and the possibility keeps floating around. One can play science and stop an experiment with the usual "suggestions for further research" and move on. It's therefore nice to see some real research on the topic as with Didier's using a prototype. However, because typesetting is pretty much about esthetics and boundary conditions we have to face reality and that's what we hit when testing. An example is the following case:

```
.... \im {x+1}.
.... \im {x+2}.
```

In the paragraph stream we get math formulas followed by a period. However, what we really get after the '1' and '2' is a math end node, a penalty, and a (likely zero) glue or kern (depending on what we configured). This means that the period is seen as a snippet and so we get a twin here, and bumping demerits then interferes with our rather advanced

test even more test more, and test more, and test even more test more, nore test more, and test even more test even more

test even more test more, and test more, and test even more test more, re test more, and test even more test even more

test even more test more, and test more, and test even more test more, re test more, and test even more test even more

Figure 21: Twins with punctuation. First paragraph has default processing; second with an experimental engine feature, third with Lua.

math spacing and penalty model. This made us be more strict in what makes for a possible sibling: we expect glue and glyph after and/or glyph and glue before. Maybe we should be even more restrictive and look at character properties which makes us end up in Lua.

Another challenge is shown in figure 21, where we have twins that are followed by punctuation. So how do we tackle that? At the Lua end we have access to the font properties so there we can act on the original Unicode character being punctuation, in which case we can ignore it. At the TEX end we need to figure that out differently. We could look at the `\sfcode` but that's rather unreliable. We could have a callback that gives the required property information, but do we really want an extra callback? In the example the third paragraph is done by our Lua implementation. The second one comes from the engine where we use an experimental character control feature that we set up for this case.[9] The verdict is still open if we add this feature, also because for it to be useful yet another field in the glyph node would be required.

So, as we move on and test more, additional constraints can occur. It is easy to come up with various "TEX should do this or that", or even "I looked into it and it can be done", and then end up with "Sorry, not now." It does take time and effort indeed but it also brings one into unknown territory. So, we do show that it can be done but we will never claim that what we do is perfect and we definitely do not enable it by default. It will take some time and likely input from ConTEXt users to fine-tune this, assuming it gets used. It can currently be enabled by setting one of the align options:

```
\setupalign[notwins]
% for the brave: [notwins,notoddlers,noorphans]
```

Let's end with some statistics. In this document we enable multiple par passes, but the number of times that these are needed is small. The extra overhead can often be neglected anyway. Here's how many first, second and emergency passes we have and how often additional sub-passes were needed to fit the criteria. In the King James we bumped the demerits by 7500 for 665 left twins, 772 right twins and 113 of these end up left and right.

context	first	second	emergency	sub-pass
page	35989	4733 (13%)	0 (0%)	282 (1%)
vbox	2942	734 (25%)	0 (0%)	0 (0%)

The document has 246,470 words, of which 112,329 get hyphenated in 35,750 checked node lists. A run without twin detection takes 14.50 seconds, with engine twin detection that gets raised to 14.75 seconds. Because here we have only text and many small paragraphs the Lua variant performs relatively slowly: 15.35 seconds. Tracing, marking words with color and reporting to the console adds .15 seconds to that. This document is not the fastest to process: we use columns, a rather demanding font, selective expansion (sub-pass driven), and the sources are XML which gets interpreted and remapped on the fly.

Thanks to Didier for inviting us to prove that it can be added to the engine with little effort and providing some stimulating statistics. Let's end with some more because it can't be that there is no performance hit when we enable this feature, right? So let's check out three scenarios:

- The `\glyphoptions` variable has the 'checktwin' bit set but both twin demerits parameters are zero, so we never enter the check.
- The `\glyphoptions` variable has the 'checktwin' bit set and both twin demerits parameters are 7500. We enter the check and per-glyph options permit it.
- The `\glyphoptions` variable has the 'checktwin' bit unset but both twin demerits parameters are 7500. We enter the check but per-glyph options prevent it from succeeding.

In ConTEXt we set the demerits and use the options bit to control it, so we always have the check but can quit after some initial tests (case 2 and 3). The numbers below are for ten runs of 15000 times each of the well-known Tufte quote, for each of the three cases:

```
\setbox\scratchbox\vbox{\samplefile{tufte}}
```

[9] Think of `\cccode"2E = "0001` (period) and `\cccode"2C = "0001` (comma) that sets the 'ignore twin' bit, where `\cccode` is the "character control" primitive.

Table 1: Results of many runs for each of the three twins cases. The last column is the average.

1	17.860	18.478	19.026	18.824	18.736	18.665	18.623	19.002	18.101	18.905	**18.622**
2	18.672	19.181	18.150	18.960	18.414	19.120	18.246	18.945	19.050	18.744	**18.748**
3	18.979	18.597	18.747	18.837	18.660	18.846	18.513	18.457	18.448	18.414	**18.650**

Table 2: Results of three runs including generating the final output, for each of the three twins cases. The first column repeats the number from table 1, and the last column is the average.

1	**no check at all**	**18.622**	37.270	37.433	37.425	**37.376**
2	**check and honored**	**18.748**	37.651	37.261	37.690	**37.534**
3	**check but ignored**	**18.650**	37.032	37.565	37.967	**37.521**

The results are in table 1. These numbers include font processing time as well as some other ConTEXt specific callback overhead processing time but we want to test with ligatures and discretionaries so this is required. When we use `\vpack` all times are the same.

But, this is for 15000 nine-line paragraphs using the Tufte quote and that is a tough one: many short words, ligatures, four hyphenated lines in the standard layout. If we output the result, we get a 3335 page document and a runtime of about 37.5 seconds (on my 2018 laptop); the numbers are in table 2.

So, in the end, assuming that we have the third variant as default (which is the most practical in ConTEXt) users will see a small performance hit due to this new feature but on a regular run, which in practice does way more than just outputting text only, no one will notice it. So, our and Didier's conclusion that we have no performance hit (something that is always considered when making a possible extension to a core component) holds.

was a queer relief, at all events—I mean for myself in especial—in the renouncement of one pretension. If so much had sprung to the surface, I scarce put it too strongly in saying that what had perhaps sprung highest was the absurdity of our prolonging the fiction that I had anything more to teach him. It sufficiently stuck out that, by tacit little tricks in which even more than myself he carried out the care for my dignity, I had had to appeal to him to let me off straining to meet him on the ground of his true capacity. He had at any rate his freedom now; I was never to touch it again; as I had amply shown, moreover, when, on his joining me in the schoolroom the previous night, I had uttered, on the subject of the interval just concluded, neither challenge nor hint. I had too much, from this moment, my other ideas. Yet when he at last arrived, the difficulty of applying them, the accumulations of my problem, were brought straight home to me by the beautiful little presence on which what had occurred had as yet, for the eye, dropped neither stain nor shadow.

Figure 22: Coloring the twins from figure 19.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University

you (as a daddy) are never too old to learn from young kids are you you (as a daddy) are never too old to learn from young kids are you you (as a daddy) are never too old to learn from young kids are you you (as a daddy) are never too old to learn from young kids are you you (as a daddy) are never too old to learn from young kids are you you (as a daddy) are never too old to learn from young kids are you you (as a daddy) are never too old to learn from young kids are you you (as a daddy) are never too old to learn from young kids are you you (as a daddy) are never too old to learn from young kids are you you (as a daddy) are never too old to learn from young kids are you

Figure 23: Final example.

SQLTEX

Oscar van Eijk

Abstract

SQLTEX is a preprocessor for LATEX files to dynamically generate documents using content from databases. All SQL code can be embedded in a LATEX file, which will be rewritten by SQLTEX for regular LATEX processing.

1 Introduction

A normal LATEX document is the basis for using SQLTEX. Using commands of the form `\sql`*cmd*, a database can be opened, data can be read from a database and placed in the output document or stored for processing in a loop, updates can be written to the database, etc. The "`sql`" command prefix can be changed via the configuration file (just like the commands themselves; we'll have a look at that further down), but will be used in the examples in this article.

SQLTEX is part of the TEX Live distributions and can be downloaded from CTAN mirrors [1]. Support is available via GitHub [2].

In the sections below you'll read a small introduction to the possibilities provided by SQLTEX. Extended information is available in the user documentation [3].

2 Configuration

SQLTEX comes with a configuration file where almost everything can be configured. Not just the environment (e.g. which database driver to use) or behavior (e.g. is it allowed to execute external programs from within SQLTEX), but even the commands can be changed. Let's have a look at that example.

By default, the command for opening a database in the `.tex` input file is `\sqldb` and has the syntax:

```
\sqldb[user=user,passwd=?,host=host]
    {database_name}
```

In this example, SQLTEX will prompt for a password.

With a few changes in the configuration file, this command can be renamed. This is useful if SQLTEX commands conflict with commands that have been defined using `\newcommand`.
For the example above, if the following lines:

```
cmd_prefix  = sql
sql_open    = db
```

are changed to:

```
cmd_prefix  = db
sql_open    = open
```

the command to open the database will be:

```
\dbopen[user=user,passwd=?,host=host]
    {database_name}
```

Multiple configuration files can be created and selected with the `--configfile` command-line option, although this option may be disabled by the system administrator as described in the "Installation" section in [3].

3 Replacement file

Data as read directly from the database might contain special characters for LATEX, like the dollar sign (`$`), curly brackets (`{}`) etc. To handle this, SQLTEX provides a replacement file with which certain characters or strings as read from the database can be replaced by their LATEX equivalent.

The syntax is straightforward: each line contains a value that should be replaced, one or more tab (→) characters (*not spaces!*) followed by the replacement value:

```
$→\$
_→\_
%→\%
&→\&
...
 →\hspace{1em}
...
```

Regular expressions are also supported. The syntax for this is `re(`*expression*`)`. For example:

```
re(<h1.*?>)→\section{
re(<h2.*?>)→\subsection{
re(<h3.*?>)→\subsubsection{
re(</h\d>)→}
```

Lines with a semicolon as the *first* character are comments.

4 The SQLTEX input file

Example use case: We have a webshop built with WooCommerce (an open-source e-commerce plugin for WordPress; see `woocommerce.com/`). We want to be able to create a printed version of certain parts of the webshop contents: for each category in our webshop it must be possible to generate a document.

4.1 Open the database

In the section *Configuration* above, we've already seen the `\sqldb` command. This requires a database name and optionally, username (`user`), password (`passwd`) and database server (`host`) can be given. The username and password can be a question mark which will cause SQLTEX to prompt for them. The default host is `localhost`.

To override the optional arguments from the command line, the options `--username`, `--password` and `--sqlserver` can be used.

doi.org/10.47397/tb/45-3/tb141eijk-sqltex

Suppose we want to connect to the database `mywpdb` on the local host with the current user and prompt for a password. The setup of our SQLTEX input file will look like this:

```
\documentclass[a4paper]{article}
\sqldb[passwd=?]{mywpdb}
\begin{document}

\end{document}
```

4.2 Load the data

The next step is loading the data.

For this, the commands `\sqlfield` and `\sqlrow` are available. Each is followed by `[`*`options`*`]` and then `{`*`query`*`}`.

For our use case we will retrieve multiple rows of data, so we need `\sqlrow`. To write this data directly to the output file, the options *`fldsep`* and *`rowsep`* can be used, e.g. for a `tabular` environment:

```
\sqlrow[fldsep=&, rowsep=\\]{
select * from my_table
}
```

But we'll need to do some more processing per row, so we won't write the data directly. Instead, we'll store it in an array for later use, using the `[setarr=`*`n`*`]` option, where *`n`* is used in the loop context in next section.

We also wanted a catalog per category. Therefore we can pass the category name as a parameter to SQLTEX on the command line. This can be used as `$PAR1` in our query. Up to nine parameters can be used in this way.

For the query, let's assume our table prefix for the WordPress tables is `wpdb_`:

```
\sqlrow[setarr=0]{
select          ms.meta_value as sku
,               p.post_title as name
,               p.post_excerpt as sdesc
,               p.post_content as ldesc
,               FORMAT(
                   IFNULL(mp.meta_value, 0)
                 , 2
                ) as price
from            wpdb_posts p
left outer join wpdb_postmeta  ms
on              p.ID = ms.post_id
           and  ms.meta_key = '_sku'
left outer join wpdb_postmeta  mp
on              p.ID = mp.post_id
           and  mp.meta_key = '_price'
left outer join wpdb_term_relationships tr
on              p.ID = tr.object_id
left outer join wpdb_term_taxonomy  tt
on              tr.term_taxonomy_id
                  = tt.term_taxonomy_id
left outer join wpdb_terms   ts
on              tt.term_id = ts.term_id
where           p.post_type = 'product'
and             ts.name = '$PAR1'
order by        ms.meta_value
}
```

4.3 Loop context

Now we will create a loop in which the data will be written to the output file per row. A loop starts with `\sqlstart{`*`n`*`}` and ends with `\sqlend{}`, where *`n`* matches the array number created in the previous section. Multiple loops can be created like this, but they *cannot* be nested.

In this loop all text and LATEX code will be written to the output file for each row. The command `\sqluse` can be used to write the fields stored in the array. They are named after the field name from the query.

The code for our loop could look like this (of course the hardwired dimensions could be replaced with macros, etc.):

```
\sqlstart{0}

\hfill\textbf{Article number: \sqluse{sku}}
\rule{\textwidth}{1pt}\vspace{1mm}
{\LARGE \sqluse{name}}

\vspace{3mm}{\Large \textbf{Description}}

\vspace{3mm}
\sqluse{sdesc}

\vspace{5mm}{\Large \textbf{Details}}

\vspace{3mm}
\sqluse{ldesc}

\vspace{5mm}{\Large \textbf{Price}}

\vspace{3mm}
\textsf{\texteuro} \sqluse{price}

\newpage
\sqlend{}
```

4.4 External scripts

There's no nice catalog without images. They can be taken from the webshop, but we need a separate script for that. It's outside the scope of this article to write such a script; let's just assume it exists and is called `fetchFeaturedImage.pl`.

We can call the script with the `\sqlsystem` command. The output of the script will then be written to the output document.

This script might need a username, password, database server, database name; it must also know

for which product the image should be retrieved. In the script that magically exists in our example, we assume some command line options to pass this data. A `sku` value is expected to identify the product.

For the database connection, SQLTEX recognizes the following predefined tags:

`<SRV>` Name of the database server.

`<USR>` Username to connect to the database.

`<PWD>` Password to connect to the database.

`<DB>` Name of the database.

Any other data can be passed as fixed text or using `\sqluse{`*`fieldname`*`}`. In the end, our call to the script could look something like:

```
\sqlsystem{./fetchFeaturedImage.pl ↩
-s <SRV> -u <USR> -d <DB> ↩
-p <PWD> -a \sqluse{sku}}
```

The `\sqlsystem` command *cannot* span multiple lines. (The ↩ symbols indicate the line break is just for *TUGboat*, not present in the real LATEX source.)

4.5 Our input file

Now we've completed all the steps, and this is how our input file looks. Let's call it `catalogue.tex`. The `graphicx` package is used for the product images, which we include with `\includegraphics`:

```
\documentclass[a4paper]{article}
\sqldb[passwd=?]{mywpdb}
\usepackage{graphicx}
\begin{document}

\sqlrow[setarr=0]{
select          ms.meta_value as sku
,               p.post_title as name
,               p.post_excerpt as sdesc
,               p.post_content as ldesc
,               FORMAT(
                   IFNULL(mp.meta_value, 0)
                 , 2
                ) as price
from            wpdb_posts p
left outer join wpdb_postmeta  ms
on              p.ID = ms.post_id
           and  ms.meta_key = '_sku'
left outer join wpdb_postmeta  mp
on              p.ID = mp.post_id
           and  mp.meta_key = '_price'
left outer join wpdb_term_relationships tr
on              p.ID = tr.object_id
left outer join wpdb_term_taxonomy  tt
on              tr.term_taxonomy_id
                  = tt.term_taxonomy_id
left outer join wpdb_terms   ts
on              tt.term_id = ts.term_id
where           p.post_type = 'product'
and             ts.name = '$PAR1'
order by        ms.meta_value
}
\sqlstart{0}

\hfill\textbf{Article number: \sqluse{sku}}
\rule{\textwidth}{1pt}\vspace{1mm}
{\LARGE \sqluse{name}}

\begin{center}
\resizebox{4cm}{!}{\includegraphics{
\sqlsystem{./fetchFeaturedImage.pl
 -s <SRV> -u <USR> -d <DB>
 -p <PWD> -a \sqluse{sku}}}}
\end{center}

\vspace{3mm}{\Large \textbf{Description}}

\vspace{3mm}\noindent
\sqluse{sdesc}

\vspace{5mm}{\Large \textbf{Details}}

\vspace{3mm}
\sqluse{ldesc}

\vspace{5mm}{\Large \textbf{Price}}

\vspace{3mm}
\textsf{\texteuro} \sqluse{price}

\newpage
\sqlend{}
\end{document}
```

5 Add a replacement file

WooCommerce can use `<span>` tags for certain font settings. They will not be replaced by the default replacement file, so for this purpose we need to create our own replacement file.

What should be there depends on what the database contains. Let's add two styles for this example and a catch-all. Our replacement file — we'll call it `catalogue_r.dat` — is shown in figure 1.

Remember '→' represents the tab character.

6 Generate the LATEX file

Finally, we must generate our catalogue. For each category in our webshop, we can run SQLTEX with the same input file and another parameter to specify the category name. In the command-line options we specify the extra replacement file we created.

The first parameter is our input file; the second parameter will replace the '`$PAR1`' placeholder.

```
sqltex --replacementfile catalogue_r.dat \
       --output guitar_catalogue.tex \
       catalogue.tex Guitars
```

```
re(<span .*?style=.*?font-size:\s*x-small.*?>)→\footnotesize{
re(<span .*?style=.*?font-weight:\s*normal.*?>)→\textsl{
re(<span .*?style=.*?text-decoration:\s*underline.*?>)→\underline{
re(<span.*?>)→{
</span>→}
```

Figure 1: Custom replacement file to handle WooCommerce `<span>` tags. The → indicates a tab character.

Article number: 138.25734

Ibanez Artstar AS113

Description

Timeless sound wrapped in a timeless finish, this semi-acoustic classic has way more to offer than you might think. The Artstar Series Ibanez AS113 electric archtop has a thinline flamed maple body coated in a vintage-flavoured lacquer. The maple construction shapes a clear and transparent foundation that packs plenty of presence, while the bound Macassar ebony fretboard and set-in three-piece nyatoh and maple neck lightly round off the treble definition. Equipped with an AS Artstar neck profile and a set of 22 medium frets, each polished off with the special Artstar fret-edge treatment, this model has an immediately snug and smooth playing feel.

Details

Product features
Number of frets: 22
Number of strings: 6
Body depth: thinline
Bridge type: tune-o-matic + tailpiece
Centre block: yes
Pickup configuration: HH
Pickup type: humbucker (passive)
Suitable style of music: warm blues, rock
Neck connection: set neck
Wood type: maple
Fretboard wood: ebony
Weight and dimensions including packaging
Weight (incl. packaging): 5.0 kg
Dimensions (incl. packaging): 120.0 x 48.0 x 19.0 cm

Price

€875.00

Article number: 138.94302

Fender Custom Shop 1963 Stratocaster

Description

1963 is a special year in the Stratocaster legend and, by some, is considered the pinnacle of Fender design. Here, the Fender Custom Shop presents an ode to that golden age in the form of the Fender Custom Shop 1963 Stratocaster NOS electric guitar. This reissued icon features a nitrocellulose new-old-stock finish in singular Shell Pink, while the body has been shaped from two specially selected chunks of alder and combined with a stunning AAA flamed maple neck. To really sweeten the deal, Fender has included a hardshell case, a strap and Certificate of Authenticity.

Details

Product features
Number of frets: 21
Number of strings: 6
Bridge type: vintage-style tremolo
Pickup configuration: SSS
Pickup type: humbucker (passive)
Suitable style of music: bright blues, country, funk, pop
Neck connection: bolt on
Body material: alder
Fretboard wood: rosewood
Weight and dimensions including packaging
Weight (incl. packaging): 10.5 kg
Dimensions (incl. packaging): 112.0 x 44.0 x 17.0 cm

Price

€4750.00

Figure 2: Two pages of the guitar catalogue generated by our use case.

Mission accomplished! The result can be seen in figure 2.

SQLTEX offers many more options, such as generating multiple documents, *if-endif* control blocks and writing updates to the database. That's all covered in the SQLTEX user documentation. The current version of the package is 3.0.

References

[1] SQLTEX on CTAN. `ctan.org/pkg/sqltex`.
[2] SQLTEX on GitHub. `github.com/oveas/sqltex`.
[3] O. van Eijk. SQLTEX v3.0. *User documentation*. `mirrors.ctan.org/support/sqltex/doc/SQLTeX.pdf`.

◇ Oscar van Eijk
Hechtel-Eksel
Belgium
`oscar (at) oveas dot com`

Styling Stata graphics with LaTeX

Travis Stenborg

Abstract

Stata is statistical software with extensive graphical data presentation tools. This article examines visual harmonization of Stata graphics for LaTeX documents. Typeface preservation guidance for exported image files is given. Supplementing Stata's capabilities with LaTeX and Ti*k*Z, such as for typographical edge cases, is also demonstrated.

1 Stata and LaTeX

Stata is commercial, general-purpose statistical software, somewhat akin to SAS and SPSS. Efforts to integrate Stata and LaTeX are not new [1, 2, 3, 4, 5, 6, 7, 9, 11, 12, 13, 16, 20]. The `texdoc` utility is especially notable for dynamically generating LaTeX documents from Stata, incorporating statistical results and graphs [8].

Styling of Stata graphics in LaTeX documents is explored here. Concluding links are provided to relevant Stata do-files, LaTeX source code, and supporting material.

2 Stata graphics

In Stata, *graphics* typically refers to graphs and plots. An illustrative example in Bayesian linear regression is included here (Figure 1). Specifically, a traceplot of one chain of Markov chain Monte Carlo (MCMC) sampling in Stata was produced. The regression analyzed the relationship between the logarithm of water catchment areas and water quality, as measured by the index of biological integrity (see e.g., [10]).

Test data were imported into Stata from the `waterQuality` dataset of the R package `ecostats`. That package accompanies an ecological statistics textbook [19], which provided confirmatory (albeit non-Bayesian) regression results.

3 Visual harmony

StataNow 18.5, MP-Parallel, quad-core edition, running on Windows 11, was used to produce the example traceplot. For visually harmonious inclusion in a LaTeX document, assuming a default Computer Modern font [15], the Stata interface was programmatically configured to render graphs in CMU Serif (a Computer Modern font family, previously installed in Windows). *Visual harmony* here refers to avoiding a blatant 'font salad' across a document and its constituent graphical labels.

Rendering in the Stata interface proceeded without issue. Export of the traceplot, however, was problematic. Formatting was retained on export to PNG and JPG raster images, and to an SVG vector image. Typeface customization was lost, however, on export to the compound image formats EPS, PS and PDF. Output reverted to Stata's default display font in those cases.

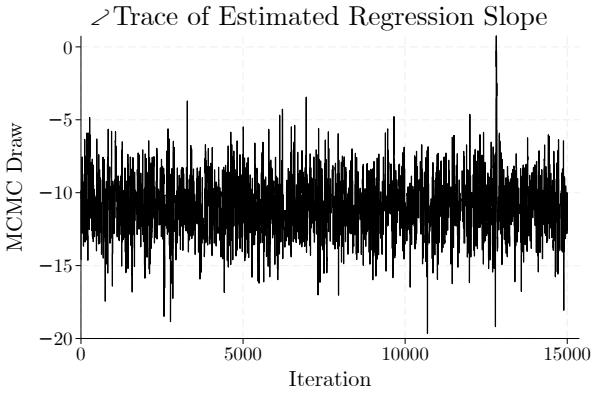


Figure 1: Example Stata MCMC traceplot for regression slope estimation in a water quality dataset. The decorative glyph prepending the title, unavailable as a Unicode or Stata Markup and Control Language glyph, was overlaid via LaTeX and Ti*k*Z.

A properly formatted traceplot, exported as a PDF file, was especially sought. A local Inkscape [14] installation was leveraged for typeface preservation. Instead of exporting directly from Stata to a PDF file, Stata's `shell` function was used to convert a correctly exported SVG file (`traceplot.svg`) to a PDF file, via Inkscape's command line mode.

```
shell "C:\inkscape\bin\inkscape.com"
   --export-type="pdf" "C:\traceplot.svg"
```

4 Supplementing Stata with LaTeX

Stata can't directly process LaTeX commands. Fine-grained suppression of graphical text elements (e.g., titles, axis and tick mark labels) is possible, however. LaTeX counterparts can then be overlaid. The approach taken for this was to create a LaTeX document using the `standalone` class, include a target image file (PDF) in a Ti*k*Z node, overlay LaTeX markup as another Ti*k*Z node, then typeset it all into a single replacement PDF file.

```
\documentclass{standalone}
\usepackage{metsymb}
\usepackage{tikz}
\begin{document}
\begin{tikzpicture}
\node[inner sep=0pt] (plot) at (0, 0)
   {\includegraphics{traceplot}};
\node[inner sep=0pt] (tag) at (-5.10, 4.36)
   {{\Large \chVI}};
\end{tikzpicture}
\end{document}
```


doi.org/10.47397/tb/45-3/tb141stenborg-stata

That process was used for the example Stata traceplot in Figure 1. The figure has a meteorological symbol, `\chVI` from the `metsymb` package [18], applied as a decorative prefix to the title. Stata supports both Unicode and a comprehensive internal symbol set (available via Stata Markup and Control Language tags [17]). In the event of gaps in glyph support, LaTeX markup can potentially address those edge cases, as in the `metsymb` example here. Overlaid complex mathematical typesetting is another potential use for supplementing Stata with LaTeX.

5 Conclusion

This material was intended to serve two purposes. Firstly, it aims to help Stata users integrate their graphical work with LaTeX. It also provides any *TUGboat* readers considering tools for statistical work with a feel for what Stata and LaTeX graphical integration would involve.

Source code and other files supporting the Stata styling described here are available on GitHub at `github.com/tstenborg/LaTeX-Styling-Stata`.

Acknowledgements

This work was supported by the Australian Research Council Training Centre in Data Analytics for Resources and Environments (project ICI9010031).

References

[1] R.J.A. Camara. Reports and other PDF documents. *The Stata Journal*, 14(1):103–118, 2014. doi.org/10.1177/1536867X1401400108

[2] Y. Chen, Y. Lian. Browse and cite Stata manuals easily: The wwwhelp command. *The Stata Journal*, 24(1):161–168, 2024. doi.org/10.1177/1536867X241233676

[3] O. de Bari, M. Himmelmann. A brief report on the first GuIT meeting. *TUGboat* 25(2):223–223, 2004. tug.org/TUGboat/tb25-2/tb81guit.pdf

[4] J.L. Gallup. A new system for formatting estimation tables. *The Stata Journal*, 12(1):3–28, 2012. doi.org/10.1177/1536867X1201200102

[5] E.F. Haghish. Rethinking literate programming in statistics. *The Stata Journal*, 16(4):938–963, 2016. doi.org/10.1177/1536867X1601600408

[6] E.F. Haghish. markdoc: Literate programming in Stata. *The Stata Journal*, 16(4):964–988, 2016. doi.org/10.1177/1536867X1601600409

[7] E.F. Haghish. On the importance of syntax coloring for teaching statistics. *The Stata Journal*, 19(1):83–86, 2019. doi.org/10.1177/1536867X19830892

[8] B. Jann. Creating LaTeX documents from within Stata using texdoc. *The Stata Journal*, 16(2):245–263, 2016. doi.org/10.1177/1536867X1601600201

[9] M. Kustudic, B. Niu. Stata tip 155: How to perform high-frequency event studies. *The Stata Journal*, 24(2):362–368, 2024. doi.org/10.1177/1536867X241258013

[10] C.A. Mebane, T.R. Maret, R.M. Hughes. An Index of Biological Integrity (IBI) for Pacific Northwest Rivers. *Transactions of the American Fisheries Society*, 132(2):239–261, 2003.

[11] L.F. Mori. Tables in LaTeX 2_ε: Packages and methods. *The PracTEX Journal*, (1), 2007. tug.org/pracjourn/2007-1/mori

[12] R.B. Newson. From resultssets to resultstables in Stata. *The Stata Journal*, 12(2):191–213, 2012. doi.org/10.1177/1536867X1201200203

[13] G. Rodríguez. Literate data analysis with Stata and Markdown. *The Stata Journal*, 17(3):600–618, 2017. doi.org/10.1177/1536867X1701700304

[14] C. Rogers. *Design Made Easy with Inkscape*. Packt, 2023.

[15] W. Schmidt. Font selection in LaTeX: The most frequently asked questions. *TUGboat* 28(2):241–242, 2007. tug.org/TUGboat/tb28-2/tb89schmidt.pdf

[16] T.W. Soon, S.L. Saw. Incorporating Stata-created PostScript files into TeX/LaTeX documents. *Stata Technical Bulletin*, 15:7–12, Sept. 1993.

[17] StataCorp, College Station, TX. *Stata 18 Graphics Reference Manual*, 2023. www.stata.com/bookstore/graphics-reference-manual/

[18] F.P. Vogt. The `metsymb` package. Generate (vectorial) meteorological symbols. ctan.org/pkg/metsymb

[19] D.I. Warton. *Eco-Stats: Data Analysis in Ecology*. Methods in Statistical Ecology. Springer, 2022.

[20] J.D. Wolfe, S. Bauldry. Collecting and organizing Stata graphs. *The Stata Journal*, 14(4):965–974, 2014. doi.org/10.1177/1536867X1401400415

◇ Travis Stenborg
Sydney, Australia
ORCID 0000-0002-2693-9628

Primo from a developer's perspective

Jan Vaněk, Hàn Thế Thành

Abstract

Primo is a cloud-based authoring, submission, and proofing tool with collaborative editing. Its target audience is authors who need to publish papers with a publisher. It simplifies the publishing process for authors and minimizes the difference between the submitted and published versions. Primo provides high-quality PDF output with math, using TEX as a back-end. Authors are shielded from the underlying XML required by the publisher as much as possible. Simply put, it's like Google Docs, but for academic publishing, with an integrated publishing workflow where the PDF preview is always accessible.

We will first provide an overview of the technologies used, the runtime components, and the structure of the source code. After that, we'll focus on one specific aspect: collaborative editing. We'll describe the parts of the document model that are crucial for collaboration.

Screenshots

Screenshots of Primo in action are shown on the next page (Figures 1 and 2).

1 Technology and runtime

Primo is written in Scala 2, which compiles to JVM bytecode and provides full interoperability with Java (code and libraries). Thanks to `Scala.js`, it also compiles to JavaScript. This allows us to share the codebase between the server and client parts. Most Primo modules have three parts: the "jvm" part, the "js" part, and the "shared" part. We use IntelliJ IDEA for development, Git for version control, and `sbt` as the build system (integrated into IntelliJ IDEA).

We use some external libraries, but not many. We created our own UI component framework, called VDL, which is used also for the internal components inside the editor. VDL is an abbreviation for "viděl", which is the past tense of "see" in Czech; it's only a name. When a Primo page is displayed, it's all VDL, except for the PDF viewer. We use SQLite for database access, Undertow for the HTTP server, Apache FTPServer for FTP connectivity, and Lucene for full-text indexing. On the client side, we use `scalajs-dom` for DOM access, along with several other libraries for both the server and client side.

At runtime, the main Primo server handles HTTPS connections from client browsers. Behind the main server, we have a variable number of so-called "remote work servers" (RWS), which are started on demand and connect to the main server via HTTP to request tasks. These RWS servers are internal and not directly accessible from outside. They are technically clients of the main server, as they initiate the connection (figure 3).

The main Primo server maintains active client sessions and opened documents, while RWS servers are stateless. They request tasks such as PDF compilation, MathML image creation, and XML validation, helping distribute CPU-heavy tasks. In the future, we might also distribute memory load across "document work servers" (DWS); these will be visible from the Internet, but hidden in the browser through the use of nested frames. This helps avoid exposing session details and complex urls to users.

2 Software structure

Primo provides several applications, such as Primo Drive and Primo Editor, each corresponding to a top-level module. Naturally, there are many submodules in the implementation. As mentioned, most modules are split into three parts: "jvm", "js", and "shared".

The source code consists of about 3,000 Scala files (6MB in total) and approximately 6,000 Git commits.

3 Document model

The document model is a tree of nodes, each with an id. There can be references between nodes, technically making it a graph, though at its core, it's still a tree. There are four types of containment:

- A node with multiple dynamic sub-nodes.
- A node with a single dynamic sub-node.
- A node with a fixed set of pre-created child nodes.
- A special node containing text, called TextPart.

Nodes can have attributes, called facets, which are atomic in that they can only be set or removed as a whole. These facets can be of any type, but there is no sub-addressing to facet values.

There are many concrete types of nodes, all derived from base node traits. The document's structure strictly follows the publisher's DTD, so we have classes that correspond directly to DTD tags, although they contain no code.

We initially tried to define our own generic set of nodes to map to the publisher's DTD, but this wasn't feasible. The DTD is too specific and large to be generalized.

Porting Primo to another DTD would require creating specific nodes, views, and functionality for that DTD. However, the core of the system remains the same, and collaborative editing is independent of these specific nodes.

doi.org/10.47397/tb/45-3/tb141vanek-primo

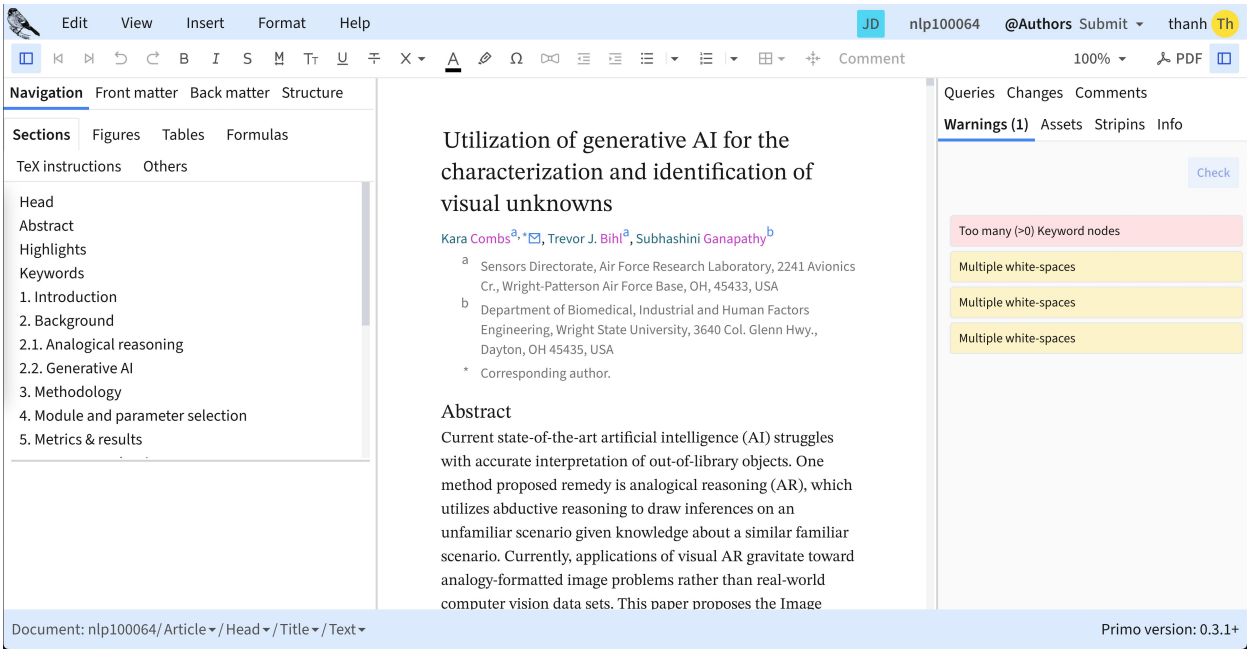


Figure 1: Primo with structural editing panel and proofing panel active

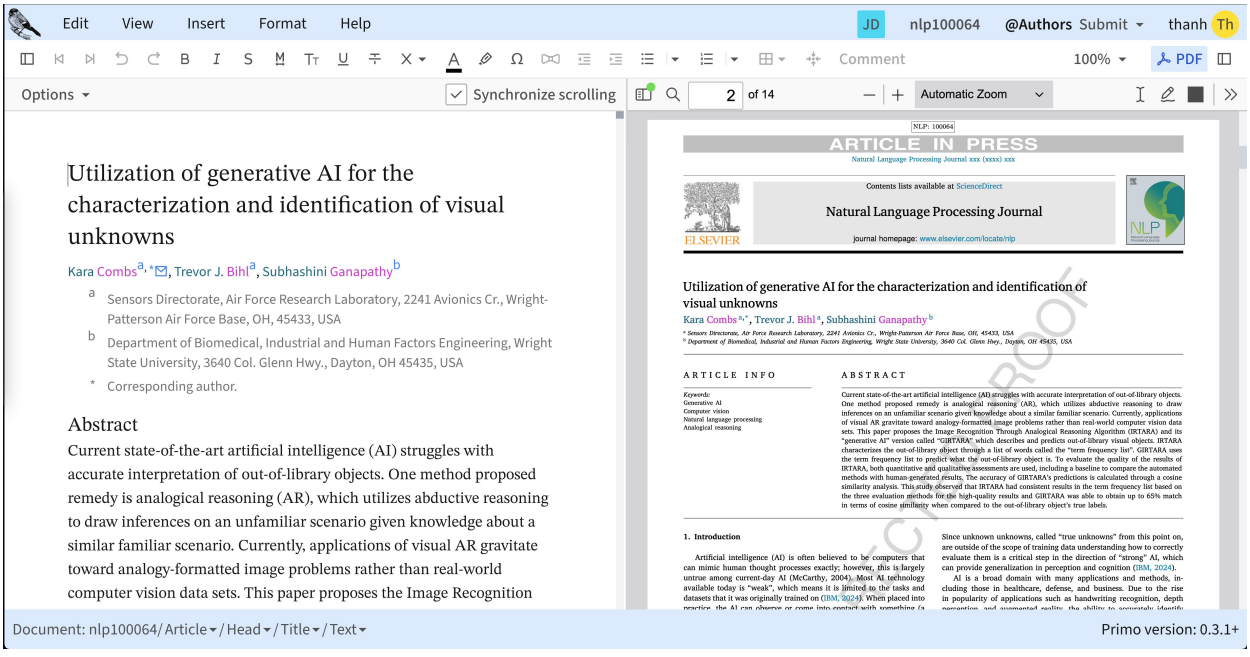


Figure 2: Primo with PDF panel active

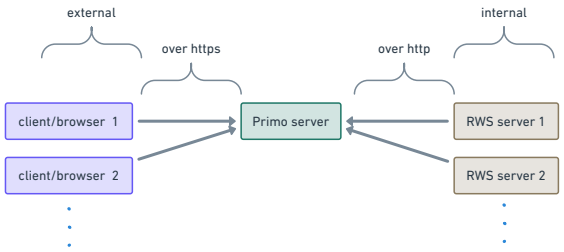


Figure 3: Communication between parties

The document model supports observation; i.e., an observer can be registered and called when the sub-node changes in the node-with-sub-node, or when sub-node is inserted or deleted from the node-with-sub-nodes, etc. The model also supports "deep-change" observation, meaning it is possible to register an observer which is called when anything anywhere below that node changes. However, while the normal observers receive information about what happened, the "deep-change" observers are only told that something happened (changed). Nevertheless, it has proven to be a useful feature for us.

Observers are also used to bind the model to the UI, ensuring that the UI updates when the model changes. This is a standard feature, so we won't go into further detail here.

In order to implement collaboration, the document model tree is internally "triplicated". There is another dimension, which we call TriLayer. The TriLayer is an enumeration of three layers:

- L0: The main layer;
- L1: The server layer;
- L2: The temporary layer.

For example, in a NodeWithSubNode, there is a sub-node for each of these layers. Similarly, in a NodeWithSubNodes, there are three arrays of sub-nodes, one for each layer. The same applies to facets and parent node references.

Here's how it might look in Scala:

```
trait NodeWithSubNode[T <: Node]
  extends ParentNode {
  private var mSubNode0: T = _
  private var mSubNode1: T = _
  private var mSubNode2: T = _
  ...
}
```

Similarly, the NodeWithSubNodes contains three references to arrays of nodes. As you might imagine, this means the model could theoretically be "triplicated", leading to three times the memory consumption. In practice, this doesn't happen. The differences in the layers exist only temporarily, until the edits are propagated to and from the server, and

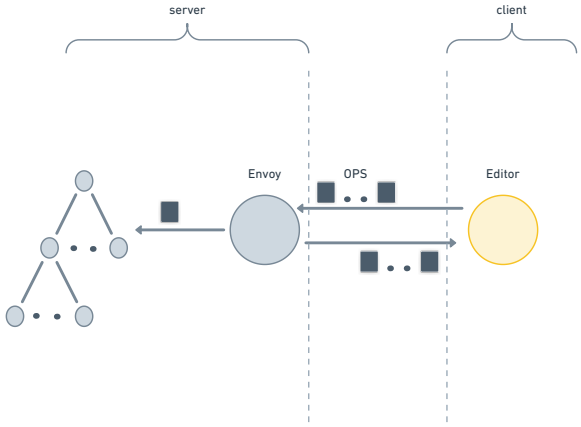


Figure 4: Client fetches changes from server

the model eventually converges so that all layers are the same. This is the state when there are no new edits. In that case, the memory overhead is just having three identical references (three memory slots) in each node (whether it is NodeWithSubNode or NodeWithSubNodes).

If we needed to, we could eliminate even that small overhead. We could have one reference (memory slot), which would either hold the single real reference or point to a special class (say TriRef), which would store the three references. After edits are processed and the model converges back to a single layer, there would be no TriRef instances, and memory consumption would be exactly the same as with a single layer. We didn't find this level of optimization necessary.

The methods that operate on the model, such as those that get or set sub-nodes in NodeWithSubNode, all take one extra parameter: the layer (of type TriLayer). This parameter is "implicit", which is a great Scala language feature. Thanks to this, we don't have to pass it explicitly; the code which uses these methods in fact looks like "normal" code, as if there were no layers. This makes the implementation much simpler.

4 How collaboration works

When a document is edited, changes are represented as operations (e.g., Insert, Delete) that are sent asynchronously to and from the server. Even when no edits are made, the client fetches the latest changes from the server asynchronously (figure 4).

When a user makes a change, operations are sent to the server. If other collaborators have made changes, those operations are fetched from the server and user's operations must be "rebased" against these new operations from the server. After rebasing, the

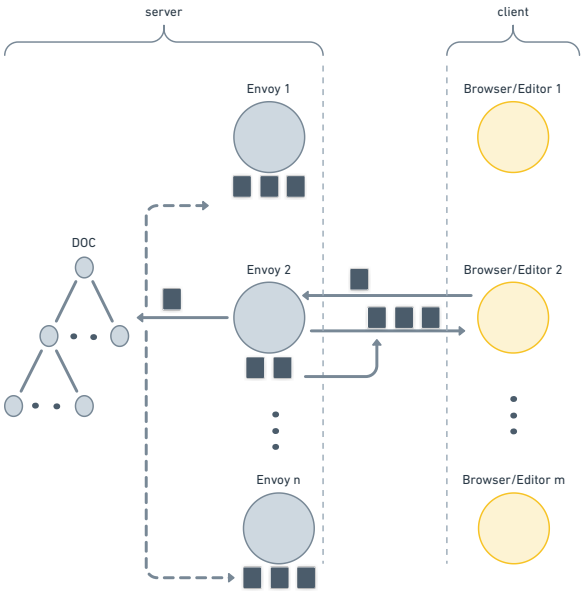


Figure 5: Rebasing local operations against operations from server

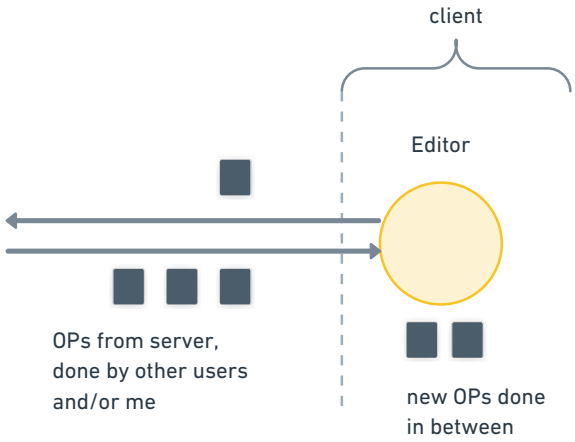


Figure 6: Operations from servers vs. new local changes in between

user's operations are applied to the document and queued to be sent to other collaborators (figure 5).

We won't go into the details of how operations are rebased against each other; it's not overly complicated.

When the client fetches operations (made by others) from the server, it needs to apply them to the document model. However, by that time, we may have made new edits and applied new operations, resulting from these edits, to the document model — operations that haven't been sent to the server yet. At the same time, our UI, which observes changes in the document model, has already been updated.

The changes fetched from the server have already been applied to the server's document model, which is considered the "higher" truth. This means we need to treat those changes as having happened before our own, unsent changes (figure 6).

How do we handle this? This is where Primo uses TriLayer(s). Of course, there are other ways to implement collaboration, but this is the approach we chose.

Primo stores the latest server version of the document in the L1 (server) layer. Any changes fetched from the server are applied to L1. Then, L1 is transferred to the L2 (temporary) layer, our own new operations are rebased against the latest operations from the server and applied to L2. Finally, L2 is transferred to the L0 (main) layer. The UI, which only observes the main layer (L0), is updated accordingly. Our rebased operations are then queued to be sent to the server during the next communication cycle. Currently, this cycle happens every second. And that's basically how it works.

5 Conclusion

Primo is designed to simplify the academic publishing process by providing a collaborative, cloud-based platform that handles the technical details of document formatting and submission. With its focus on generating high-quality PDFs and minimizing the differences between submitted and published versions, Primo allows authors to concentrate on their content rather than technical hurdles.

This article covered the core aspects of Primo's architecture and development, with some features and technical details omitted due to the scope of the discussion. Primo's architecture is built with flexibility and scalability in mind, allowing it to adapt to future demands of the academic publishing process.

◇ Jan Vaněk
Trivic s.r.o.
jan (at) trivic dot io

◇ Hàn Thế Thành
Trivic s.r.o.
thanh (at) trivic dot io

GNU Emacs and AUCTEX on Windows and macOS for the rest of us

Vincent Goulet

Abstract

This short note introduces *Emacs Modified for Windows* and *Emacs Modified for macOS*, easy-to-install distributions of GNU Emacs that come bundled with packages for LaTeX users and R developers.

1 Introduction

In *TUGboat* 45:1, Arash Esbati of the GNU project contributed a very interesting article on building and setting up an editing environment centered on GNU Emacs on Windows [1]. As a seasoned Emacs user, I learned quite a few tricks myself.

The most technical and challenging part of the procedure proposed by Arash is the compilation of Emacs on the non-free operating system that is Windows. The purpose of this short note is to let *TUGboat* readers know of an allegedly simpler starting point.

I maintain distributions of GNU Emacs for Windows and macOS that come bundled with a few select packages for LaTeX users and R developers, most notably AUCTeX [3] and the Emacs Speaks Statistics (ESS) project [2]. In other words, the distributions provide the LaTeX editing environment out of the box.

The distributions go under the decidedly non-poetic names of *Emacs Modified for Windows* and *Emacs Modified for macOS*.[1]

2 Short history

The *Emacs Modified* project started with Windows as of around 2002 — I've lost track of the early days of the project for various reasons. My aim then was to provide my students with an Emacs fully set up for R programming and LaTeX usage, and that would be easy to install. On Windows, this means: distributed with an installation wizard.

In 2007 I officially announced the availability of *Emacs Modified for Windows* on the `ess-help` mailing list and the distribution became the recommended installation method for Windows (and later for macOS) on the ESS web site.

Around the same time, I started using macOS as my main operating system and I wanted to use a graphical version of Emacs. Aquamacs was my first stop:[2] it was well integrated with the Mac GUI and it also came with AUCTeX, ESS and a number of other packages preinstalled. Unfortunately, Aquamacs played a lot with frames and fonts behind your back, and the versions of the bundled packages eventually started lagging. In need of a more "stock" version of Emacs, I launched the *Emacs Modified for macOS* project. I was the first user, besides being the creator, of this distribution.

Emacs began shipping with its own package manager with version 29.1, released in 2023. I thought this would mean the end of the road for my project. However, a very informal survey among the ESS community showed on the contrary that there is still interest for my all-in-one distributions.

3 Features of the Windows version

Emacs Modified for Windows is a distribution based on the 64-bit (`x64`) version of Emacs made available through GNU.[3] The additions to stock Emacs are the following: AUCTeX, ESS and polymode, markdown-mode, Tabbar, a patched version of `psvn.el`, and configuration files to make everything work together. In addition, this distribution ships with the spell checker Hunspell[4] and dictionaries for English, French, German, and Spanish. Last, but not least, the software is distributed with an InnoSetup installer, making installation trivial.

It is worth noting that the installer creates a `HOME` environment variable on Windows if it does not already exist. The presence of this environment variable used to be crucial to the stability of Emacs. Although it may no longer be completely crucial, my experience is that it remains useful when working with other software coming from the Unix world.

4 Features of the macOS version

Emacs Modified for macOS is based on GNU Emacs compiled by David Caldwell.[5] The modifications are the same as the Windows version, with, in addition, Steve Purcell's `exec-path-from-shell.el` to import the user's environment at Emacs startup. Without the latter, using outside programs within Emacs is much more difficult. Furthermore, the macOS distribution does not ship with Hunspell (pre-built binaries are not readily available for the platform), but installation instructions using Homebrew and the four dictionaries mentioned above are provided.

The software is distributed as a disk image and installation proceeds as is common on macOS: by copying the Emacs image into the *Applications* folder using the Finder.

[1] `emacs-modified.gitlab.io`
[2] `aquamacs.org`
[3] `ftp.gnu.org/gnu/emacs/windows`
[4] `hunspell.github.io`
[5] `emacsformacosx.com`

doi.org/10.47397/tb/45-3/tb141goulet-emacs

5 Build procedure

The build procedure of the distributions can be summarized as follows: unpack a stock GNU Emacs distribution in a temporary directory; install the extension packages from source inside the Emacs tree; repack everything using InnoSetup on Windows or `hdiutil` on macOS. On the latter, the software also needs to be notarized by Apple. Once the latest version of all the components is determined, the whole procedure is automated using `make`.

6 Conclusion

GNU Emacs is the Text Editor of text editors. If you want to use Emacs on Windows or macOS without having to compile or install many pieces of software yourself, you may consider the distributions of the *Emacs Modified* project. Batteries are included, and no agent will visit you.

7 Acknowledgments

Many in the ESS community have helped me improve and maintain the distributions over the years. I especially wish to thank Martin Mächler (ETH Zürich) and Rodney Sparapani (Medical College of Wisconsin) for their continued encouragement and support.

References

[1] A. Esbati. Building a modern editing environment on Windows around GNU Emacs and AUCTEX. *TUGboat* 45(1):77–87, 2024. doi.org/10.47397/tb/45-1/tb139esbati-auctex

[2] A.J. Rossini, R.M. Heiberger, et al. *ESS — Emacs Speaks Statistics*. ESS Developers, 2024. ess.r-project.org

[3] K.K. Thorup, P. Abrahamsen, et al. *AUCTEX: A sophisticated TEX environment for Emacs*, 2024. www.gnu.org/software/auctex/manual/auctex.index.html

◇ Vincent Goulet
École d'actuariat, Université Laval
Québec, Canada
vincent.goulet (at) act dot ulaval dot ca
https://vigou3.gitlab.io

Making PDF text unreadable

Hans Hagen

Nothing you put on the web is safe any more from being scraped, used and abused by these large language models ran by Big Tech. It has the potential not only to kill creativity but might also discourage original authors to make something public. It is for this reason that ConTEXt now provides a way to prevent such abuse. Take the following example:

> Many readers will skim over formulas on their first reading of your exposition. Therefore, your sentences should flow smoothly when all but the simplest formulas are replaced by "blah" or some other grunting noise.

In order to protect this precious thought of Donald Knuth, we can do this:

```
% one emoji
\enabledirectives[backend.pdf.nounicode=×]
```

which gives, when we cut and paste from the PDF:

We can be a bit more visual when we do:

```
% two emoji
\enabledirectives[backend.pdf.nounicode=×✓]
```

Maybe experimental quantum computers can untangle the result:

If needed we can challenge these models a bit:

```
\enabledirectives[backend.pdf.nounicode=noai]
```

because the distribution of character usage in a language gives away some information:

```
nooi iioaiin iooo nioi ooii ioiiioon
oo ooioi oino iioaoon oi ioii inaonoa
oooo. ooiiiioii, ioii niooionin nooioa
ooi nioooooi ioio ooo nio ooi noia
aoino ioiiioon oii iiaoonia ni “nooo”
oi noii oooii niioooon oooni.
```

When we target Microsoft we can say:

```
\enabledirectives[backend.pdf.nounicode=nopilot]
```

and wonder if its artificial text cruncher is able to combine this semi-random output into something that makes sense, assuming that a user expects that. In a worst case scenario it might recall your constantly stored editing state and use that.

```
nooi iponpii ooii iooo oopi ooionioi
oo ltpoi piil iponooo oo ioni plooiol
looo. itpipooip, ioni ipolpoipi itonin
ooo ioooltii otpo oii onl ltp iool
oipil ooionioi oip ipoioipn oi “oiot”
oi ioop oltpi oinolooo oooip.
```

We leave it to the reader to come up with a nice word to confuse the Adobe analyzers. In case you wonder what happens here: we just generate a random `/ToUnicode` vector for the fonts used. And, because in ConTEXt we subset fonts using a sparse index one can get (somewhat) back to the real Unicode characters only when additional resources in the PDF are consulted, which (we bet) will not be done.

◇ Hans Hagen
Pragma ADE

doi.org/10.47397/tb/45-3/tb141hagen-inaccessible

\topmark in output routines without \shipout

Udo Wermuth

Abstract

Developers of formats for TEX sometimes need to write their own output routine that replaces the macro \plainoutput of plain TEX. One reason for a new output routine might be to do checks for a page without directly sending it to the dvi file. This article describes how one can avoid the \topmark of the next page becoming invalid in such a scenario.

1 Introduction

It is well-known that the program TEX is accompanied by the file plain.tex that implements the format plain. The combination of program and format builds the system that the manual for TEX, *The TEXbook* [3], explains. The contents of the file is also described in detail in the manual's Appendix B. A format is required as the program without a format, INITEX, provides only a minimal starting point and needs many additional declarations and definitions to become a useful tool for the average user. The file plain.tex delivers a set of more-or-less necessary additions.

The format plain provides not only a complete initialization of TEX, it also includes macros that help users write common parts of various document types. For example, plain contains the macros \beginsection and \proclaim. The first outputs a heading and starts a section for a text; the second allows the presentation of a theorem in a mathematical paper. The macros of plain are usually efficient but limited, although they can handle a wide variety of texts. For example, a book project should replace the abovementioned macros. Donald E. Knuth describes in [3, App. E] the macros that he uses for *The TEXbook* and shows on page 259 a version of \beginsection for another of his book projects.

This observation is applicable to most macros of plain, even those that a user doesn't directly call, like the \output routine: \output={\plainoutput} [3, p. 255f.] where \output is a token list required by the program TEX and \plainoutput a macro of plain. As part of a general purpose format these macros have certain limitations. Thus the TEX user groups supported in TEX's early years the exchange of enhanced macros and later CTAN started to store them for general access. And the published papers provided feedback to TEX's manual: Knuth read the *TUGboat* articles until mid-1983 as a preparation for its Appendix D; see [5, comment after entry 775].

One limitation of \plainoutput can be seen on this article's pages: It doesn't support multiple-column output; *TUGboat* provides its own output routine to get the two-column layout. (*The TEXbook* presents two methods to do that; see page 257 and pages 386–388.) Other problems show up when one has to work with spreads, i.e., the two pages that a reader sees when a book is opened. Publishers sometimes want to change the page height but keep it balanced for a spread; see [1]. (A remark about the reference: Nowadays one can query the badness of a created box via \badness and insertions can be kept using \holdinginserts [5, entries 883 and 884].)

Whenever \plainoutput is invoked it produces a page for the dvi file so that TEX discards the processed data. But TEX allows the output routine to return this data to the input stream for reprocessing. This behaviour is named here a *silent output path* if TEX starts it because of a negative penalty.

Contents. Section 2 gives a brief overview about output routines without discussing the complex timing aspects and the associated lists. Section 3 lists a few use cases for reprocessing input. Section 4 discusses silent output paths and points out a problem with such output routines. Sections 5, 6, and 7 describe methods for how the problem can be solved.

2 Selected aspects of output routines

The program TEX checks at several occasions, for example, at the end of a paragraph or after \output has ended [3, p. 122], if so much material was collected that the best page break (according to a *cost function* [3, p. 111]) was found. Usually that means more material than the page needs is available; otherwise the break might be forced because TEX found a penalty ≤ -10000. (Note, when TEX finishes a very long paragraph it might have material for several pages; see also [3, ex. 14.15].) TEX keeps all unused material for later processing. It is preceded by a \penalty10000 if the break was at a penalty item. In this case TEX stores the penalty of the breakpoint in the parameter \outputpenalty; otherwise \outputpenalty gets the value 10000 [3, p. 125].

As any penalty ≤ -10000 has the same effect, "reasons" for the call of \output can be coded with the values -10001, -10002, ... [3, p. 400]; plain uses the value -20000 to signal a \supereject that outputs all insertions that are on hold; see [3, pp. 254 and 353]. (-1073741824 is also special [3, p. 264].)

Box 255 and \shipout. Earlier the program TEX removed from the top of the collected material all glue, kern, and penalty items and later it moves the

doi.org/10.47397/tb/45-3/tb141wermuth-silent

page's main text to \box255 without complaining about under- or overfull boxes [3, pp. 112 and 400]. \box255 must be empty when communication with the output routine, the token list stored in \output, starts. TEX fills \box255 and the output routine must empty it again; \plainoutput does this via an \unvbox255 [3, p. 120] inside a new vbox. In \plainoutput, a headline and footline are added and insertions are placed before this vbox is sent to the dvi file by the command \shipout [3, p. 254].

TEX's integer parameter \maxdeadcycles determines after how many calls to \output it requires a \shipout. (INITEX sets \maxdeadcycles = 25.) Every time \output is called TEX increases another parameter, \deadcycles, and resets its value to 0 after a \shipout. TEX raises an error and ships out the page itself if \deadcycles > \maxdeadcycles. In this case the program TEX executes something like "\output={\shipout\box255}" [3, p. 253].

Insertions and marks. The user's \output routine receives not only \box255, which contains the text for the page, but also all boxes from the defined insertion classes [3, p. 122]. The data from each insertion class is cleared when it is transferred to its associated box for \output. This describes TEX's default behaviour. The insertions remain in \box255 if \holdinginserts > 0 [3, p. 125].

\topmark receives the data from the \botmark of the previous \box255. \firstmark and \botmark are filled with the first and last mark of the current \box255. Without a \mark in this box, TEX fills them with the contents of \topmark; see [3, p. 258].

Page data. With \tracingpages > 0 TEX documents the calculations of the cost function in the log [3, p. 112]. The trace shows the goal height and the current total together with the maximal stretch and shrink values for the page so far. One can inspect these values as TEX makes them available as dimensions in pt, called \pagegoal, \pagetotal (the current height), \pagedepth (the current depth), \pagestretch, \pagefilstretch (in pt not fil), \pagefillstretch (in pt), \pagefilllstretch (in pt), and \pageshrink (see [3, pp. 114]). Some of them are used by plain's \midinsert [3, p. 116] to check if the insertion fits on the current page or if it must be transformed into a delayed \topinsert.

\pagegoal is initialized with \vsize and decreased by insertions, i.e., when \insert is used. In plain, \insert is called by \vfootnote, which is called by \footnote, and by \endinsert if it closes a \topinsert or a transformed \midinsert.

Final remark. All actions inside \output are executed inside a group that the TEX program adds. Therefore, everything is a local change unless we prefix it by \global [3, p. 21].

3 Some reasons for a delayed \shipout

Let's state two observations.

1. The plain format has only one insertion class for illustrations and figures: \topins [3, p. 363]; the \endinsert macro uses it. As the name indicates, the insertion is placed at the top of the page if there is enough room. Otherwise it is placed at the top of the next page that has enough space for this insertion after all previous topinserts have been placed. That is, the sequence of the insertions as they occur in the document is kept. This is not the case for plain's \midinsert, as it isn't handled as an insertion class. And a new insertion class, say \botins, would keep its own sequence independent of \topins. Thus, insertions might not appear in the sequence in which they occur in the input.

2. Figures 1 and 2 of [9] show that (a) the move of a club line from an odd-numbered page to the next and (b) the move of an insertion from a left page to the preceding spread improve the layout.

Therefore people want to control the placement of insertions themselves. Doing that automatically might best be handled in the output routine, with \holdinginserts = 0: Every time TEX finds an insertion it is forced to call the output routine that decides about the placement even if there is not enough material for a complete page yet; that is, there will be no \shipout. As the input must not be lost but boxes must be emptied, the insertion is temporarily saved and in a later call to \output reprocessed.

Others want to change and balance the page height in a spread [1] or try to avoid widow and club (or orphan) lines at a page break [2], for example by changing the number of lines in a paragraph. Again, people look for automatic solutions of such problems and work with output routines that have paths without a \shipout.

4 Silent output paths

Page 254 of *The TEXbook* shows a do-nothing output routine. (I added the comments.)

```
\output={%a do-nothing output routine, TB p.254
 \unvbox255 %        take page out of its \vbox
 \ifnum\outputpenalty<10000 %      and reinsert
    \penalty\outputpenalty    % this penalty if
 \fi}% there was a page break at a penalty item
```

The \unvbox255 returns \box255's contents to the input for reprocessing. If no parameters change, the same \box255 is built when this or any other output routine is called. But we don't have a chance to call another output routine if TEX executes the do-

nothing output routine. (A reader who wants to test that will quickly get the abovementioned error message "Output loop—25 consecutive dead cycles".) And remember every call to `\output` now adds a `\penalty10000` waiting to be processed by TEX. (Insert `\showlists` at the start of the do-nothing output routine to see this penalty item in your test.) Moreover, with `\holdinginserts` = 0 insertions are lost from the input, i.e., they survive in their boxes but TEX does not change `\pagegoal` anymore for an `\output` with `\shipout`.

A silent output path. We want to call the silent output path in vertical mode via a unique negative penalty < -10000 and $\neq -20000$ that can be tested and deleted in `\output`. As the do-nothing routine only wastes resources let's add something useful. The `\box255` is the last box that TEX creates before it fires up the `\output` routine; see [4, §1012 and §1017]. Thus, its badness is stored in `\badness` and can be accessed; see the remark in section 1.

```
\output={\ifnum\outputpenalty=-"CBAD %  = -52141
  \immediate\write16{Badness of page so far:
   \the\badness}\unvbox255
 \else\immediate\write16{Badness of completed
   page: \the\badness}\plainoutput \fi}
```

The silent path of this output routine is triggered by a unique negative penalty: `-"CBAD`. To avoid TEX's inserted `\penalty10000` cancelling page breaks, a penalty item should be used. It also ships out all completed pages so that only an unfinished page is left when the silent output path starts its work. Together with setting `\holdinginserts`, we put `\penalty0` into a macro; one can also move it out of the macro and then use other values < 10000. The macro must be used only in vertical mode.

```
\def\dosilentoutput{% trigger silent output path
 \edef\curmode{\ifinner-\fi\ifvmode 1 \else
  \ifhmode 2 \else\ifmmode 3 \else 0 \fi\fi\fi}%
 \ifnum\curmode=1 {\penalty0 % annihilates TeX's
  % \penalty10000; ships out all completed pages
  \holdinginserts=1 \penalty-"CBAD }%
 \else\errhelp{Call \dosilentoutput only in non%
   -internal vertical mode. I'm forced to ignore
   it.}\errmessage{Vertical mode required}\fi}
```

A side effect of silent output paths. In 1988 a LATEX user wrote about a *mysterious phenomenon*: "... when the marginal note instruction is added, the value of `\topmark` gets trashed." Leslie Lamport answered [8]: "Not so mysterious... LATEX processes figures and marginal notes by calling the `\output` routine (by inserting a vertical penalty less than -10000). (This is done to enable LATEX to figure out whether the marginal note will be on an even- or odd-numbered page.) Of course, the value of `\topmark` gets destroyed in the process." (See also this newsletter (2022) `https://ctan.org/macros/latex/base/ltnews35.pdf` for new developments.)

The do-nothing output routine does nothing, as required, but the silent output path might destroy something if `\box255` contains a `\mark` command. It is correct that *The TEXbook* writes that everything in `\box255` is returned for a second processing by TEX. But in section 2 it was stated that there is a link between the current page and the previous `\box255` via `\topmark`. When TEX builds a new `\box255` for a silent output path there might be a new token list for its `\botmark`, which becomes the `\topmark` for the next page, i.e., the next `\box255`.

An example. Let me illustrate the problem with an example. There are three pages and on each page there is a `\mark` containing the page number. Assume that a silent output path is triggered on page 3. Either the text contains the `\mark` after the silent output path was called or TEX saw the `\mark` when `\box255` is built for the silent output path. Thus we have two cases:

page	mark	`\topmark`	`\firstmark`	`\botmark`
[1]	1	null	1	1
[2]	2	1	2	2
3 silent	–/3	2	2/3	2/3
[3]	3	2/3	3	3

and only the first case behaves as if the silent output path was not called. In the second case we see that the final third page does not have the correct `\topmark`, i.e., the contents of page 2's `\botmark`.

Use the following code together with the above macro `\dosilentoutput` and the stated `\output` to see the problem for `\topmark`.

```
\headline{\hfil\tenrm   % show marks in headline
 \topmark---\firstmark---\botmark\hfil}
% three pages (with 2 lines) each with one \mark
\vsize=\topskip \advance\vsize by \baselineskip
One\par \mark{1} Two\par                % page 1
Three\par \mark{2} Four\par             % page 2
Five\par \mark{3}\dosilentoutput Six\bye% page 3
```

Exchange `\mark{3}` and `\dosilentoutput` to get the other case.

Final remark. As stated in section 2, at some point TEX throws away vertical skips, kerns, and penalties at the beginning of a page. Of course, a page should not begin, for example, with a vertical skip as we usually want to have the first lines of the pages at the same distance from the top edge of the paper. But this cancellation is not recoverable; whatever is thrown away at this place is gone forever. (See the conversation between Fred Bartlett, Donald E. Knuth, and Peter Breitenlohner

on page 20 of [7].) Thus, call a silent output path only at places where nothing is lost or where the loss of the mentioned items doesn't create a problem.

5 Keeping \topmark valid

There is only one way to get a valid \topmark after a silent output path is executed: A page must be built that contains the required mark contents as its last mark, i.e., as its \botmark. This must happen just before the page that has to be shipped out next is generated. Sure, the complete page that was shipped out last could be saved and reprocessed but this time without shipping it out so that everything is rebuilt. But actually only its \botmark is required.

To implement the above description we define three macros. The first stores the \botmark of a shipped-out page. As mentioned above a \global assignment is required as TEX executes \output inside an (invisible) group.

```
\newbox\PrevPageBotMark
\def\SaveCurrentBotMark{% save the \botmark of a
% normal output routine, i.e., one with \shipout
 \global\setbox\PrevPageBotMark% global required
  =\vbox{Restore\mark{\botmark}\vfil}}
\SaveCurrentBotMark %initialize \PrevPageBotMark
```

(Note that "Restore" or anything else that starts horizontal mode is required to have later some material for \box255; \vfil switches to vmode.)

The second macro must be called to finish all silent output paths. It creates a new page by unboxing the last stored \botmark. Next, it triggers another call to \output via a unique negative penalty ≤ -10000.

```
\def\CloseSilent{% create page with the \botmark
% for silent output routine to keep the \topmark
 \unvcopy\PrevPageBotMark \penalty-"C0FF
 \unvbox255 }
```

The third macro realizes the task of the second silent output path. It has to clear \box255 that contains the newly created page of the above macro. As we want neither to ship out the page nor to unbox the page for reprocessing, \box255 is unboxed into another box: I take \box0. This assignment is local and \box0 is restored when \output ends.

```
\def\RestoreTopMark{% empty \box255, do not ship
% it out and do not put it back for reprocessing
 \setbox0=\vbox{\unvbox255 }}
```

A corrected \output. The token list of the above \output gets a new case that tests the parameter \outputpenalty against −"C0FF of the second macro.

```
\output={% this routine keeps the valid \topmark
 \ifnum\outputpenalty=-"CBAD %          = -52141
  \immediate\write16{Badness of page so far:
  \the\badness}\CloseSilent
 \else\ifnum\outputpenalty=-"C0FF %     = -49407
  \RestoreTopMark
 \else\SaveCurrentBotMark
  \immediate\write16{Badness of completed page:
  \the\badness}\plainoutput
 \fi\fi}
```

What happens with our example from the previous section? Now the table has one more line as there are two calls to silent output paths.

page	mark	\topmark	\firstmark	\botmark
[1]	1	null	1	1
[2]	2	1	2	2
3 silent 1	–/3	2	2/3	2/3
3 silent 2	2	2/3	2	2
[3]	3	2	3	3

Independent of whether there is a \mark on the page when the first silent output path gets called, the \topmark of the shipped-out page 3 is correct.

6 Using more than one output routine

In the previous section the silent output path was placed together with the default output routine inside the token list of \output. When we want to look at more than one page the silent output path should get its own \output. In this section we want to determine the net height of the box with the material that TEX has collected for the next pages including insertions. (Not included are the stretch and shrink components of skips as well as the extra space of insertions ([3, p. 122]).) Thus, the value is different from \pagetotal that looks at a single page only.

Extended default. The first \output is based on plain's default. It must be extended by the procedure that reestablishes the \topmark. Again, we call this procedure with a negative penalty: −"C0DA. As this is the point when the default output routine takes back control we also reset \holdinginserts.

```
\output={% plain \TeX's output routine, extended
% by two macros to keep \topmark valid
 \ifnum\outputpenalty=-"C0DA %          = -49370
  \RestoreTopMark \global\holdinginserts=0
 \else \SaveCurrentBotMark \plainoutput \fi}
```

We keep the declaration of \PrevPageBotMark, perform its initialization, and also reuse the two macros \SaveCurrentBotMark and \RestoreTopMark of the previous section.

Silent \output. The second output routine is activated through a macro. This silent output path is triggered by the negative penalty −"C0DE. Then it displays the height of the box with all material that is waiting to be processed by TEX for the next pages. We store the material in a box called \SaveBoxCclv.

```
\def\SilentOutput{% create a silent \output
 \global\output={\TwoSilentPaths}}
```

The main task of the second output routine is done in the macro \TwoSilentPaths. In our example it is quite simple as it must only store material: (a) \box255 and (b) if at a page break with a penalty, that penalty (i.e., the \outputpenalty).

```
\newbox\SaveBoxCclv
\def\TwoSilentPaths{%
 \global\setbox\SaveBoxCclv   % collect material
   =\vbox{\unvbox\SaveBoxCclv \unvbox255 }
 \ifnum\outputpenalty=-"C0DE          % = -49374
  \immediate\write16{height of box with the
   collected material: \the\ht\SaveBoxCclv}
  \SwitchOutputRoutine  % back to default output
 \else\ifnum\outputpenalty<10000 %    break at a
  \global\setbox\SaveBoxCclv = \vbox{%   penalty
   \unvbox\SaveBoxCclv \penalty\outputpenalty}
 \fi\fi}
```

Note that we need to replace \CloseSilent as the original output routine must be restored. And we need to unvbox not only \PrevPageBotMark but also the new box \SaveBoxCclv.

```
\def\SwitchOutputRoutine{%activate stored output
 \global\output=\SaveOutput    % restore \output
 %end silent output routine and restore \topmark
 \global\SilentOutputActivefalse
 \unvcopy\PrevPageBotMark \penalty-"C0DA
 \unvbox\SaveBoxCclv\penalty0 }% return material
```

Activation. The last thing we have to do is to replace the macro \dosilentoutput. We need to store the current output routine in a new token list. Next we have to activate the above silent output path. Moreover, as we want to apply the silent output path to, for example, a very long paragraph we must split the macro. Otherwise the end of the paragraph triggers the extended output routine before the activation macro takes control. When two macros require to be called in sequence we should use error messages if a user calls them in the wrong order.

```
\newtoks\SaveOutput % save current \output
\newif\ifSilentOutputActive % changed \output?
\def\beginsilentoutput{% start the silent output
 \ifSilentOutputActive
  \errhelp{You already said \beginsilentoutput
   and I'm using a silent \output routine. Maybe
   an \endsilentoutput is missing. I'll forget
   that you said \beginsilentoutput again.}%
  \errmessage{Silent output routine is active}%
 \else \global\SilentOutputActivetrue
  \global\holdinginserts=1 %        keep inserts
  \global\SaveOutput=\output \SilentOutput
 \fi}
```

The macro \endsilentoutput should only be called in vertical mode. Thus we write two error messages.

```
\def\endsilentoutput{%  activate the task of the
 \ifSilentOutputActive   % silent output routine
  \ifvmode \penalty-"C0DE
  \else \errhelp{Call \endsilentoutput only in
    vertical mode. I'm forced to ignore it.}%
   \errmessage{Vertical mode required}\fi
 \else\errhelp{You didn't say \beginsilentoutput
   so I cannot end a silent \output routine. Go
   on, nothing bad can happen.}\errmessage{No
   silent output routine active}\fi}
\let\goodbye=\bye  % forgotten \endsilentoutput?
\outer\def\bye{\ifSilentOutputActive \par
  \immediate\write16{(end stops silent output)}%
  \penalty-"C0DE
 \fi \csname goodbye\endcsname}
```

The last four lines prevent the output-loop error (see section 4) if \endsilentoutput was forgotten in the input.

An example. Let's take the example of section 4 and change it a little bit so that it becomes an example for this section.

```
\headline{\hfil\tenrm   % show marks in headline
 \topmark---\firstmark---\botmark\hfil}
% three pages (with 2 lines) each with one \mark
\vsize=\topskip \advance\vsize by \baselineskip
One\par \mark{1} Two\par                % page 1
Three\par \mark{2} Four\par             % page 2
\beginsilentoutput Five\par \mark{3}
\endsilentoutput Six\bye                % page 3
```

As expected all topmarks are correct. The message reports 32 pt as we collect the text of page 2 (22 pt), which was not yet output by TEX when it sees the macro \beginsilentoutput, and the first line of page 3 (10 pt).

Some remarks. In this section the prefix \global was applied in front of \holdinginserts (twice), \output (twice), and \SaveOutput. This allows, for example, to code \beginsilentoutput in a paragraph — together with \endsilentoutput after the \par — to call the silent output routine for this paragraph.

As before we use a \penalty0 (see the macro \SwitchOutputRoutine) to defeat TEX's inserted \penalty10000. The contents of the shipped-out pages, shown with \tracingoutput=2, contain pairs of the form (..\penalty 0, ..\penalty 10000) at places where the silent output path ends. For instance, page 3 of the above example shows such a pair where \endsilentoutput was called.

7 Combining both methods

The method of section 5 works for the last unfinished page, the one of section 6 for any material even if

TEX outputs it in more than one page. Here we show how one can use both methods in one document.

The sections start with different \output routines; the one of section 6 is the minimal requirement and we start with it although this makes the macros more complex as the macro \dosilentoutput of section 4 must activate the one of section 5.

```
\newif\ifOutputBadness
\def\ActivateBadnessReport{% for \dosilentoutput
\ifSilentOutputActive %("CBAD) in output routine
\else \OutputBadnesstrue % start report \badness
 \output={%this routine keeps the valid \topmark
  \ifnum\outputpenalty=-"CBAD %         = -52141
   \immediate\write16{Badness of page so far:
    \the\badness}\CloseSilent
  \else\ifnum\outputpenalty=-"C0DA %    = -49370
    \RestoreTopMark \global\holdinginserts=0
  \else \SaveCurrentBotMark
   \immediate\write16{Badness of completed page:
    \the\badness}\plainoutput
 \fi\fi}\fi}
```

It's a combination of both output routines, for example, the test of \outputpenalty against −"C0FF is gone and only the test −"C0DA is used. It does not make sense to keep two identical tests. Note, we do not perform the activation between the macros \beginsilentoutput and \endsilentoutput since they change \output. To use \dosilentoutput between these macros, call \ActivateBadnessReport earlier than the macro \beginsilentoutput.

```
\def\dosilentoutput{%  silent \badness reporting
 \ifvmode %with preparation: both vertical modes
  \ifOutputBadness\else\ActivateBadnessReport\fi
  \ifOutputBadness \penalty0
   \global\holdinginserts=1 \penalty-"CBAD
  \else \errhelp{Call \ActivateBadnessReport
    before \beginsilentoutput/ \endsilentoutput
    if \dosilentoutput is used between them.}%
   \errmessage{Cannot activate output routine}%
  \fi
 \else \errhelp{Call \dosilentoutput only in
   vertical mode. I'll forget that I saw it.}%
  \errmessage{Vertical mode required}\fi}
```

Next, we must change \CloseSilent of section 5 as it uses −"C0FF. Moreover, in the case that it is executed between \beginsilentoutput and \endsilentoutput we cannot restore the box with the saved \botmark as it becomes a part of \SaveBoxCclv. Luckily, we don't need to do that.

```
\def\CloseSilent{% create page with the \botmark
% for silent output routine to keep the \topmark
 \ifSilentOutputActive % except if called inside
 \else \unvcopy\PrevPageBotMark % another silent
 \fi \penalty-"C0DA \unvbox255 }% output routine
```

The macro \SilentOutput must be extended to cover the silent output path of section 5.

```
\def\SilentOutput{% create a silent \output
 \global\output={\ifnum\outputpenalty=-"CBAD
   \immediate\write16{Badness of page so far:
    \the\badness}\CloseSilent
  \else \TwoSilentPaths
  \fi}}
```

Acknowledgments. The topic "silent output path" occurred in a discussion between Max Chernoff, Karl Berry, and me. At the end there was an agreement that the results, although probably not new, should be documented to make them better known. I accepted the task to write a few pages for *TUGboat*. I used the contributions of the other two in this text without giving explicit credit. Careful proofreading by Karl Berry and Barbara Beeton improved readability.

References

[1] Frederick H. Bartlett, "Automatic Page Balancing Macros Wanted", *TUGboat* **9**:1, 1988, 83. tug.org/TUGboat/tb09-1/tb20bartlett.pdf

[2] Max Chernoff, "Automatically removing widows and orphans with lua-widow-control", *TUGboat* **43**:1, 2022, 28–39. tug.org/TUGboat/tb43-1/tb133chernoff-widows.pdf
Update in *TUGboat* **43**:3, 2022, 340–342. tug.org/TUGboat/tb43-3/tb135chernoff-lwc.pdf

[3] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[4] Donald E. Knuth, *TEX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[5] Donald E. Knuth, "The Errors of TEX", *Software — Practice and Experience* **19** (1989), 607–685; reprinted as Chapters 10 and 11 in [6], 243–339. The log, i.e., Chapter 11, is still updated: ctan.org/tex-archive/systems/knuth/dist/errata/errorlog.tex

[6] Donald E. Knuth, *Literate Programming*, CSLI Lecture Note No. 27, 1992. www-cs-faculty.stanford.edu/~knuth/lp.html

[7] Donald E. Knuth, "TUG'95: Questions and Answers", *TUGboat* **17**:1 (1996), 7–22. tug.org/TUGboat/tb17-1/tb50knut.pdf

[8] Leslie Lamport, "LaTeX Notes", *TeXhax Digest V88*:11, 1988-02-01; in: ctan.org/tex-archive/info/digests/texhax/88/texhax.11.gz

[9] Michael F. Plass, *Optimal Pagination Techniques for Automatic Typesetting Systems*, Ph.D. thesis, Stanford University, 1981, 6+72 pages. tug.org/docs/plass/plass-thesis.pdf

◇ Udo Wermuth
Dietzenbach, Germany
u dot wermuth (at) icloud dot com

About TEX's error messages

Udo Wermuth

Abstract

Some people state that they don't like TEX's error messages; some are even frustrated. This article looks at TEX's error messages and tries to make a more objective assessment by checking them against, among other things, results from the research field *Human-Computer Interaction*.

1 Introduction

To err is human and humans most often understand each other even if there are errors in their communication. Computer programs that interact with humans are different: Every typo, every forgotten but required symbol throws an error message. Sometimes an error message is unable to identify the problem correctly and gives a frustrated user seemingly unrelated information.

On the *TEX – LATEX StackExchange* pages there is a question on the so-called *TEX meta* site with the name "TEX Community Polls" (`https://tex.meta.stackexchange.com/questions/1564/tex-community-polls`). There users can ask questions about TEX that have purely opinion-based answers. One of the questions is "What do you dislike in (LA)TEX?" The three top ranked answers as of April 2024 are "Bad error messages" (165 votes), "Packages that conflict with each other" (60 votes), and "(LA)TEX syntax and programming style" (58 votes). Thus there is a clear winner.

Of course, such a poll cannot claim to be representative. For example, one aspect is that only 3894 users are listed on the meta site although there are more than 250,000 on the main site.

The tag "errors" that is attached to questions on the main site which treat compilation and similar errors is the twenty-first most used tag as of April 2024: Approximately 2% of all questions on this site carry this tag. It doesn't seem that the primitives of any TEX engine or the format `plain` trigger many questions about compilation errors. Only 6 questions are listed for the join of the two tags "errors" and "plain-tex" and the combination "errors" and "tex-core" reports just 52 questions, which is less than 1% of all questions with the tag "errors".

To find an explanation of these two observations doesn't seem to be easy as it looks as if they send contradictory signals: TEX's error messages are "bad" but there are only a few "problems" with them that are worth asking a question.

Moreover, we don't know how the users work with TEX. Do they enter texts, program macros, or write mathematics? The format of the displayed error messages is always the same (see section 2) but the user needs more information and skills in cases two and three. Next, the TEX engine isn't the only source of error messages. Formats like plain TEX or LATEX add their own error messages as well as macro packages that are used with these formats. We will look in a later section at the primitives of the original TEX program that allow formats and macro packages to output error messages in the style of this program. But we don't look at the error messages issued by formats or macro packages. This article concentrates on the error messages that are coded in the original TEX program.

TEX is a sophisticated program. It puts a high value on what can be achieved with its features if a skilled person utilizes TEX in the right way. Even if a beginner only wants to enter text using plain TEX to create a "masterpiece of fine printing" [3, p. 28] TEX's documentation must be studied. For example, only after 34 pages of *The TEXbook* [3] — without the dangerous-bend paragraphs ≈ 24 pages — does the first-time reader know enough to be guided to create a simple document. Unfortunately some users do not follow this advice. The StackExchange main site has questions that show that for some users the difference between warning and error message is unknown, terms like *underfull hbox* or *overfull hbox* are unfamiliar, and the black rectangle at the end of an overfull line is a surprise. Such a user might not be able to understand TEX's error messages independent of their quality. But even if a beginner studies the documentation one expected fact remains: Beginners need more time to understand an error message and their fix is more often wrong compared to experienced users. *Human-Computer Interaction* (HCI) research supports this statement; see [7].

That does not mean that error messages are a problem only for beginners. Experts have more experience and are able to identify the root cause of the error more quickly. They aren't confused by a bad error message as they've probably seen and debugged it before. Or as stated in [7]:

> Yet, ask any experienced programmer about the quality of error messages in their programming environments, and you will often get an embarrassed laugh. In every environment, a mature programmer can usually point to at least a handful of "favorite" bad error responses. When they find out that the same environment is being used by novices, their laugh often hardens.

doi.org/10.47397/tb/45-3/tb141wermuth-errors

We look at only one program, the original TEX that is started from a command line. Some editors or online services implement additional features that should help to handle error messages but this article concentrates on TEX's error messages as displayed by the program itself.

Is there a way to get an objective assessment about the quality of TEX's error messages? Well, HCI research contains the subtopic *error messages*; see [1] for an overview. Unfortunately, the empirical studies sometimes find conflicting evidence about how useful certain techniques are. For example, [1] states that it's not clear if highlighting of error messages helps to make them more understandable.

We can also look at the recommendations that practitioners formulate or have to follow because of standards. For example, ISO 9241:110 *Ergonomics of human-system interaction, Part 110: Interaction principles* contains a section "Use error robustness".

In some sense it is unfair to confront software from 1982 with modern insights about user interaction. But TEX's beginners experience these modern insights in other often-used software. Thus, they judge TEX's error messages according to what they are used to. And it might not be unfair as TEX wasn't designed only for programmers.

Contents. Section 2 gives an introduction to TEX's error messages. Section 3 looks at HCI insights and lists 14 recommendations that are used to evaluate TEX's error messages in section 4. Section 5 discusses some peculiarities of TEX's reporting. Section 6 checks TEX's user manual. The last section contains a summary and some final remarks.

2 Error messages in the program TEX

I counted for TEX 111 error message *templates* (not counting complete duplicates, "`\errmessage`", and "Interruption"; see below). Most of them are for a single error message but TEX varies the wording in some of them. For example, the template "You can't use '⟨command⟩' in ⟨mode⟩" is used with several commands and modes; two examples are: "You can't use '`\moveleft`' in horizontal mode" and "You can't use '`\spacefactor`' in vertical mode".

All but one error message comes in two parts. The first part consists of the official error message. The second part is an associated help message. Some messages have more than one help text and some help texts are used for more than one error message. As we saw in the previous paragraph the error message might contain command names and technical terms. For the help text Knuth allows more than one line and writes in a different style. From [5, §72]:

> . . . these informal helps should use simple vocabulary that complements the words used in the official error message that was printed. Outside the U.S.A., the help messages should preferably be translated into the local vernacular.

TEX displays all error messages in the same format but sometimes with additional lines if more information must be shown to the user. Except for "runaway errors" (see section 5) the first statement is the official error message preceded by an exclamation mark and a space; it might contain a line break. The last two lines split the line (or its relevant part) from the user's input into the input that TEX has read and the input that TEX hasn't looked at yet. Such two-line output is called a *context* by TEX; a context always consists of exactly two lines. The first line is preceded by "l.*n*" and a space if the context refers to an input file; *n* is the line number of the currently active input file. (A fact that isn't known to all users of TEX, as questions on the StackExchange main site show.) The second line is indented to start after the first line's end; it might be empty if TEX has read the line completely. Let's look at an example: Assume `\moveleft` is used in horizontal mode on line 5 of the input file. TEX stops and displays:

```
! You can't use `\moveleft' in horizontal mode.
l.5 A\moveleft
                B
?
```

The ?-prompt is TEX's way to start the interactive dialog with the user [3, p. 31]. Entering 'h' or 'H' and ⟨return⟩ makes TEX output the help text.

```
? h
Sorry, but I'm not programmed to handle this case;
I'll just pretend that you didn't ask for it. If
you're in the wrong mode, you might be able to return
to the right one by typing `I}' or `I$' or `I\par'.

?
```

At the end a new prompt is shown indicating that TEX wants to get the next command from the user.

In the interactive dialog, data can be added on top of the input from the file. The above-shown help text explains that a user can give TEX additional input at the ?-prompt if it is prefixed by 'I' (or 'i'). By entering a number between 1 and 99, characters and control sequences from the input file can be skipped.

Often TEX tries to recover from a user's error and inserts something into the processing. For example, if the input file contains a symbol that is only allowed in the math modes but TEX is in horizontal mode then it tries to fix the problem itself.

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   ^
l.7 2^
      3$
? h
I've inserted a begin-math/end-math symbol since I
think you left one out. Proceed, with fingers
crossed.
```

This is an example of the above-mentioned case that TEX sometimes displays additional lines. Although TEX read '^' as shown in the line that starts with "l.7" it states that it will read this symbol again after it inserted a '$' to start inline math mode. Of course, the math mode should start in front of the '2' although in this case it doesn't make a difference. But the input "x^3$" doesn't produce the same result as "x^3" as the 'x' is in a wrong font. Section 5 shows an example where the fix fails; that's why the user should "proceed with fingers crossed".

There are also other reasons to display additional lines in an error message. If TEX is in the middle of a macro expansion then the displayed error message contains another context that shows the expanded macro. For example, "\," is a macro usable only in math mode with the expansion "\mskip \thinmuskip". Thus, the input "\,x^3$" generates the following error message with one more context.

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   \mskip
\,->\mskip
          \thinmuskip
l.3 \,
      x^3$
?
```

A user who enters ⟨return⟩ gets an unwanted skip in front of the 'x'. It's better to remove '$', '\mskip', and '\thinmuskip' and to insert a new "$". Enter "3" and TEX skips three *tokens*, i.e., characters or control sequences. Although the term "token" is used in the help menu that's shown with a "?" at the ?-prompt, only a dangerous-bend paragraph defines it [3, p. 38]. After "3" and ⟨return⟩ TEX displays:

```
\,->\mskip \thinmuskip

l.3 \,
      x^3$
```

TEX skips the inserted-text '$' and as the macro context shows the to-be-read-again '\mskip', and the '\thinmuskip'. Next we start math mode with "I$" at the ?-prompt. (With a single context the correct fix is easier to detect than for an error in a deeply nested macro when TEX shows many contexts.)

Sometimes TEX fixes a problem and doesn't allow the user to change the fix. For example, a missing number in front of a unit in the assignment to a dimen register is fixed by TEX by inserting zero. Thus, "\dimen0=pt" makes TEX display the following error message.

```
! Missing number, treated as zero.
<to be read again>
                   p
l.1 \dimen0=p
             t
? h
A number should have been here; I inserted '0'. (If
you can't figure out why I needed to see a number,
look up 'weird error' in the index to The TeXbook.)
```

The help text shows that there are more complicated error situations in which it is not obvious why a number is missing. We learn what this means in a later section.

Please note that the "to be read again" information is a context, i.e., it might show text on two lines. For example, the input "\dimen0=1trupt" with a missing 'e' in the keyword "true" makes TEX display the following error message.

```
! Illegal unit of measure (pt inserted).
<to be read again>
                   t
<to be read again> t
                    ru
<to be read again>
                   p
l.2 \dimen0=1trup
                 t
? h
Dimensions can be in units of em, ex, in, pt, pc, cm,
mm, dd, cc, bp, or sp; but yours is a new one! I'll
assume that you meant to say pt, for printer's
points. To recover gracefully from this error, it's
best to delete the erroneous units; e.g., type '2' to
delete two letters. (See Chapter 27 of The TeXbook.)
```

The first context means that TEX wants to go back to the beginning of what should be the unit, the second that it needs to read 'ru' after it read 't' again, and the third that it must read the beginning of the keyword for the real unit again. TEX detected that "true" wasn't complete not earlier than it found the 'p'. Of course, in this case one has to enter "5", not "2" as stated in the help text, to remove the "trupt".

By default TEX shows five contexts. If the error is deeply nested in a sequence of macros this number might be too small. It can be increased by

assigning a larger number to TEX's integer parameter `\errorcontextlines`. Omitted contexts are indicated by TEX with three dots if the parameter is larger than zero. The official error message, the first context (above it was either "inserted text" or "to be read again"), and the context for the input are always displayed.

Warning messages. Besides error messages the program TEX knows several warning messages. In essence there are five templates with a lot of cases.

1. Missing character: There is no ⟨*x*⟩ in font ⟨*font name*⟩!
2. *3 options*: Underfull | Loose | Tight
   2 opt.: `\hbox` | `\vbox`
   (badness ⟨*b*⟩) has occurred while `\output` is active in
   2 opt.: paragraph | alignment
   2 opt.: at lines ⟨*x*⟩–⟨*y*⟩ | detected at line ⟨*z*⟩
3. Overfull
   2 opt.: `\hbox` (⟨*m*⟩ pt too wide) | `\vbox` (⟨*m*⟩ pt too high)
   has occurred while `\output` is active in
   2 opt.: paragraph | alignment
   2 opt.: at lines ⟨*x*⟩–⟨*y*⟩ | detected at line ⟨*z*⟩
4. (end occurred inside a group of level ⟨*n*⟩)
5. (end occurred when ⟨`\if...`⟩ on line ⟨*l*⟩ was incomplete)

Item 1 is a single case in which the character (or slot number) ⟨*x*⟩ and the ⟨*font name*⟩ are variables. But item 2 has $3 \times 2 \times 2 \times 2 = 24$ cases: One of the three words from the first line can be combined with one of the two words from the second line, then with one of the two words from the second-last line, and next with one of the two phrases from the last line. Of course, the badness value ⟨*b*⟩ and the line numbers ⟨*x*⟩, ⟨*y*⟩, ⟨*z*⟩ are variables too as well as ⟨*m*⟩, ⟨*n*⟩, and ⟨*l*⟩ in items 3–5. Item 3 represents eight cases and in item 5 all known `\if`-commands can occur.

Item 1 is reported in the log file if TEX's integer parameter `\tracinglostchars` is larger than zero. As the message goes only into the log file, TEX outputs at the end of the run "(see the transcript file for additional information)". In some sense the user has made an error as a character should be typeset that's not available in the font; at least the output isn't what the user had in mind. For example, this message occurs if a user wants a lowercase letter while typesetting with the font `cminch`.

Item 2 is reported on the terminal — and in the log file, like all terminal output after this file was opened — if the badness of an hbox is larger than `\hbadness` or the badness of a vbox larger than `\vbadness`. Item 3 is also shown on the terminal if for an hbox either `\hbadness` < 100 or the excess is larger than `\hfuzz`, and for a vbox either `\vbadness` < 100 or the excess is larger than `\vfuzz`. When the warnings of items 2 and 3 occur TEX wasn't able to do the typesetting with the current parameters, i.e., the user hasn't directly done anything wrong except perhaps giving a parameter an unusually small value, for example, the `\hsize`.

Items 4 and 5 are displayed by TEX on the terminal at the end of a compilation to indicate that it was able to follow the input but it detected at the end of the run that the user forgot either to close a group or to enter a `\fi`. The output might be correct by chance but usually it is not what the user wants. For example, if a users enters "`\iftrue` ⟨text without `\fi`⟩ `\bye`" TEX must typeset all of ⟨text without `\fi`⟩ before it finds `\bye`, making it display the warning message of item 5. TEX doesn't know at which place the user forgot to enter `\fi`, but it knows where the conditional starts. A similar situation occurs if `\iftrue` is replaced by, for example, `\begingroup`; then TEX displays the message of item 4 but *without* the line where the group is opened. To find this line is often not easy.

It can be confusing that TEX reacts differently in two similar situations, once with a warning (as above) and once with an error message. TEX gives an error if the user enters "`\iftrue\else` ⟨text without `\fi`⟩ `\bye`":

```
! Incomplete \iftrue; all text was ignored after
line 9.
<inserted text>
                \fi
<to be read again>
                   \bye
l.9 \iftrue\else \bye

? h
A forbidden control sequence occurred in skipped
text. This kind of error happens when you say
‘\if...’ and forget the matching ‘\fi’. I’ve
inserted a ‘\fi’; this might work.
```

The reason for the error message is described in the help text. TEX is skipping text as the ⟨*false branch*⟩ after an `\iftrue` isn't executed by TEX. But suddenly TEX finds `\bye`, a macro defined to be `\outer`. Such a macro is forbidden in skipped text [3, p. 206]. We will look at `\outer` in a later section.

Digression: Influence of using `WEB`. Don Knuth states his programs became better only because he uses his literate programming system `WEB`. One of his arguments describes how this system improves his coding of error messages and the associated recovery

process [6, p. 126f.]. The way in which WEB programs separate the program lines that build the specified functionality from the necessary error checking of arguments and data lets the programmer pay more attention to the error checking procedures.

The program TEX is written in WEB and thus benefits from this separation. In TEX's program an early part implements a bunch of procedures for error reporting; see part 6, §§72–98, of [5]. Features of these procedures include the distinction of recoverable errors (§82) and fatal errors that require ending the current TEX run (§§93–94), an easy way to add a help text to an error message (§79), a dialog to interact with TEX if it stops after showing a message issued by the non-fatal error procedure (§§83–88), the detection of a standstill if the user doesn't want the interactive dialog (§76), and a way for the user to interrupt a compilation (§98). However, computer limits of the 1980s certainly restricted the possibilities.

All (except one) non-fatal error messages provide a help text that the user can retrieve through the interactive dialog. The help texts are also placed in the log file (§90). (The exception is an error message for experts who set up hyphenation patterns.) Knuth encourages programmers who port TEX to translate for non-English users the help messages; see above. This is allowed in the TRIP test [4, p. 3].

3 Recommendations for error messages

The conclusion of [1] — a comprehensive, historical, and state-of-the-art report, written 2019, with 219 cited articles about research done since 1965 in the area "programming error messages" — states:

> Programming error messages are important but problematic, and have been for over half of a century. Without a more coordinated effort this is unlikely to change. Currently there is not even an agreed way to measure the effectiveness of programming error messages, effectively.

It turns out that less than 4% of the cited articles analyze runtime error messages; the majority examine error messages during a static compilation.

The article presents guidelines as a kind of intersection from various research projects. No paper has stated all of them and maybe no one has used them since then. We also consult two other sources that give recommendations for error messages. Both address the needs of users who have to work with software, but who aren't programmers themselves.

Thus, I build my own mix of recommendations.

Research. Ten guidelines are presented in [1] with long descriptions. In a table they are listed as "increase readability", "reduce cognitive load", "provide context", "use a positive tone", "show examples", "show solutions or hints", "allow dynamic interaction", "provide scaffolding", "use logical argumentation", and "report errors at the right time".

I summarize the long descriptions in single sentences, accepting the risk that some details get lost.

Error messages should ...

1. ... be expressed in plain language, i.e., in familiar vocabulary without technical jargon.
2. ... not increase complexity in order to show what is wrong.
3. ... provide the correct location of the error with some context.
4. ... be formulated in a friendly and polite way avoiding words like "invalid", "illegal", etc.
5. ... show examples of similar errors.
6. ... offer solutions or give hints.
7. ... allow dynamic interaction with the user.
8. ... help users to learn and advance.
9. ... make concrete claims in a correct logical argumentation.
10. ... be reported as early as possible.

This is a long list and no single research paper that is referenced in [1] formulates all ten items. Some items are specifically meant to support beginners (for example, item 8), as many papers concentrate on novice programmers; [7] is just one of them.

Heuristics. Jakob Nielsen's "Ten usability heuristics" for user interface design [8] addresses error messages too. He states:

> Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Thus, the heuristics focus on items 1, 3, and 6 from the above list.

ISO 9241. Nowadays part 110 of this voluminous standard deals with error messages. I don't use the latest revision but a much older one from 1996 when part 110 was still called part 10. This standard was renamed in 1996 and revised in the years 2006 and 2020 to extend it for all interactive systems including hardware.

I use the German standard [2] from 1996, in my own translation. The list below contains statements from four sections: "Conformity with user expectations", "Self-descriptiveness", "Use error robustness", and "Controllability". The generic standard sometimes assumes that the user has a graphical user interface with individual *fields* for the data that has to be input; this is called the *dialog system*. The

standard allows for these requirements to be ignored if no such fields are used.

A. Messages should be displayed in an understandable, factual, constructive, and uniformly structured manner. They should not contain any value judgments, such as "absurd input".
B. The dialog system should use the vocabulary that the user is familiar with through the work.
C. The dialog system should protect the user from making mistakes. (For example, a field that expects digits must not accept letters.)
D. Errors should be explained to the user to help to fix them; this includes the location of the error, its type, and the methods of how to fix it.
E. Depending on the task, it may be desirable to make a special effort for the presentation to support the recognition of the error situation and its subsequent correction. (For example, a field is marked if the error is associated to this field.)
F. If the dialog system can correct errors automatically, it should inform the user that the correction has been made but should also give the user the opportunity to overwrite the correction.
G. User concerns and characteristics may make it necessary to postpone the handling of error situations in order to allow the user to decide when to handle them. (For example, corrections from the spell checker are only displayed on request.)
H. It is desirable to provide on request additional explanations for troubleshooting.
I. Entries should be checked for validity and correctness of the data before the entry is activated. Additional intervention options should be provided for commands with a large impact. (For example, a list of items that should be deleted is shown; the deletion is carried out only after the user has confirmed it.)
J. It should be possible to correct errors without having to switch the state of the dialog system. (For example, the incorrectly entered address of a letter can be changed without losing the content of the letter.)
K. The amount of explanations should be changeable to meet the skill level of the user.

Some aspects mentioned in this list are covered by the ten items that we listed earlier. Most of item A is stated in items 1, 4, and 9. Only the aspect "uniformly structured manner" seems to be missing, as I wouldn't put it under item 2. Item B is covered by item 1 and item D by items 3 and 6. On the other hand items C, E, G, I, and J don't seem applicable to a system like TEX as they focus on an input field. I added examples of what the standard means to explain these items in more detail. Thus only items A, F, H, and K are added to the above list of ten items:

Error messages should ...

11. ... appear in a uniformly structured manner.
12. ... contain a description of the correction that the system must perform automatically if it is able to apply one. The user must be able to overwrite this correction.
13. ... provide an additional explanation that is shown on request.
14. ... provide a customizable amount of explanation.

4 Evaluation of TEX's error messages

I looked at all 111 error messages and decided for each of the 14 recommendations of section 3 if the pair of error message and help text fulfills it. Sometimes the decision was easy as the fulfillment was clear. However, sometimes a weighing up was necessary; then I tried to be very strict.

1. use plain language. Is the name of a command "jargon"? I think it's not if the command occurs in the user's input although Nielsen's heuristics state "no code". More problematic is an error message involving an unknown control word; for example, as shown above the user typed "`\,`" and TEX responds with "`\mskip`", a command that the user might not know. But it's familiar: Every user can identify it as a TEX command. On the other hand an error message that writes "Bad metric (TFM) file" might be considered to use jargon that not every user knows.

107 error messages fulfill this recommendation.

2. don't increase complexity. This recommendation is violated if something happens that is unexpected like a "This can't happen" or a runaway error message (see section 5). The complexity is increased too if the help text refers to *The TEXbook*.

93 error messages fulfill this recommendation.

3. provide location. All error messages provide a location as seen in section 2 but sometimes it's the location at which the error is detected, not where the error is made. As explained under the subsection "warnings" in section 2 this is unavoidable.

107 error messages fulfill this recommendation.

4. friendly formulation. The error messages and the help texts are formulated in a friendly and factual way. Although I think users aren't offended, I reject "invalid" (2×), "illegal" (6×), and "improper" (7×); I accept "incomplete" (1×), "incompatible" (3×), and "bad" (9×) as in "Bad mathchar".

96 error messages fulfill this recommendation.

Table 1: Evaluation of TEX's error messages (TEX version 3.141592653)

	tex.web	Recommendations													
no.	line no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2064	✓		✓	✓						✓	✓	✓	✓	
2	2073	✓		✓	✓						✓	✓	✓	✓	
3	2094	✓		✓	✓						✓	✓	✓	✓	
4	2098	✓	✓	✓	✓							✓	✓	✓	
	2134	not evaluated: Interruption													
5	6134	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
6	6143	✓	✓	✓			✓	✓	✓	✓	✓	✓		✓	
7	7157	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓	
8	7186	✓			✓	✓	✓	✓		✓	✓	✓	✓	✓	
9	7318	✓	✓	✓			✓	✓		✓	✓	✓	✓	✓	
10	7724	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
11	7760	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
12	8070	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
13	8087	✓			✓	✓	✓	✓		✓	✓	✓	✓	✓	
14	8134	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
15	8197	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
16	8265	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
17	8393	✓		✓	✓		✓	✓		✓	✓	✓		✓	
18	8467	✓	✓	✓				✓	✓	✓	✓	✓		✓	
19	8571	✓	✓	✓	✓		✓	✓		✓	✓	✓		✓	
20	8621	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	
21	8632	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	
22	8646	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	
23	8657	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	
24	8668	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	
25	8746	✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	
26	8790	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
	8803	not evaluated: duplicate to #17													
27	8955	✓	✓	✓			✓	✓	✓	✓	✓	✓		✓	
28	8990	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	
29	9043	✓	✓	✓			✓	✓	✓	✓	✓	✓		✓	
30	9057	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓		✓	
31	9351	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
32	9365	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
33	9372	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
34	9416	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	
35	9515	✓		✓	✓			✓		✓	✓	✓	✓	✓	
36	9746	✓	✓	✓	✓			✓		✓	✓	✓	✓	✓	
37	9788	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
38	9899	✓	✓	✓	✓			✓		✓	✓	✓	✓	✓	
39	10266	✓	✓	✓	✓			✓		✓	✓	✓	✓		
	10268	not evaluated: same template as #39													
40	10955			✓	✓		✓	✓		✓	✓	✓	✓	✓	
	10956	not evaluated: same template as #40													
41	11049	✓		✓	✓		✓	✓		✓	✓	✓	✓	✓	
42	11240	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
43	11275	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
44	12750	✓	✓	✓	✓			✓		✓	✓	✓	✓	✓	
45	14234	✓			✓	✓		✓	✓	✓	✓	✓	✓	✓	
46	15359	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	
47	15468	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
48	15487	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
49	15631	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
50	16293	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	
51	18298	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	
52	18309	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
53	18741	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
54	18761	✓		✓	✓			✓			✓	✓	✓	✓	
55	18774	✓		✓	✓			✓			✓	✓	✓	✓	
56	18805	✓		✓	✓			✓			✓	✓	✓	✓	
57	19020	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
58	19073	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
59	19398	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
60	19539	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
61	19644	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓		✓	
62	19775	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
63	19916		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
64	19958	✓	✓	✓	✓			✓		✓	✓	✓	✓	✓	
65	19968	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
66	20413	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
67	20437	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
68	20699	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
69	20699	✓	✓	✓	✓			✓		✓	✓	✓	✓		
70	20722	✓	✓	✓	✓			✓		✓	✓	✓	✓	✓	
71	20737	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
72	20927	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
73	21000	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
74	21043	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
75	21166		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
76	21202	✓		✓	✓		✓	✓		✓	✓	✓		✓	
77	21341	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
78	21442	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	
79	21450	✓	✓	✓	✓			✓		✓	✓	✓	✓	✓	
80	21467	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	
81	21567	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
82	21571	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
83	21582	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
84	21608	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
85	21614	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
86	21656	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
87	21673	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
88	22029	✓		✓	✓		✓	✓		✓	✓	✓	✓	✓	
89	22058	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
90	22103	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	
91	22200	✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓	
92	22204	✓	✓	✓	✓	✓		✓		✓	✓	✓	✓	✓	
93	22289	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
94	22390	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
95	22438	✓		✓	✓			✓		✓	✓	✓	✓	✓	
	22448	not evaluated: same template as #95													
96	22476	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
97	22637	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	
98	22752	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
99	22761	✓	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	
100	22794	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
101	22931	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
102	23048	✓	✓	✓			✓	✓	✓	✓	✓	✓		✓	
103	23095	✓	✓	✓	✓			✓		✓	✓	✓	✓	✓	
104	23117	✓	✓	✓	✓			✓		✓	✓	✓	✓	✓	
105	23190	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	
106	23222	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
107	23239	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	
108	23310			✓	✓			✓	✓	✓	✓	✓	✓		
	23383	not evaluated: duplicate to #6													
109	23395	✓	✓	✓			✓	✓	✓	✓	✓	✓		✓	
	23572	not evaluated: `\errmessage`													
110	23821	✓	✓	✓	✓				✓	✓	✓	✓	✓	✓	
111	24890	✓	✓	✓	✓			✓		✓	✓	✓	✓	✓	
number:		1	2	3	4	5	6	7	8	9	10	11	12	13	14
✓ total:		107	107		35		106		104		111		108		
			93		96		81		53		110		89		0

5. show examples. No error message contains an example. But the help texts sometimes do. I accept this as fulfilling this recommendation.

35 error messages fulfill this recommendation.

6. offer solutions/hints. A solution is presented if TEX fixes the error automatically. Additionally, several help texts contain hints.

81 error messages fulfill this recommendation.

7. allow interaction. This is fulfilled by design except if the error is fatal and TEX stops immediately. This happens, for example, with "TeX capacity exceeded, sorry". (See section 5.)

106 error messages fulfill this recommendation.

8. help to learn. Often the explanation in the help text adds information that educates the user so that it's unlikely that the error is made again. For example, the help text of the error message "Incompatible magnification" states "I can handle only one magnification ratio per job". I also accept an error message that a number is out of range when the help text states the correct range.

53 error messages fulfill this recommendation.

9. argue logically. No error message argues illogically. Nevertheless, for example, fatal error messages like "Emergency stop" report facts and don't refer directly to the user's input in their statements. I reject such error messages.

104 error messages fulfill this recommendation.

10. report early. By design the error message is shown as soon as TEX detects an error. Only the message "This can't happen" that occurs without an earlier error message seems to violate this recommendation.

110 error messages fulfill this recommendation.

11. use uniform structure. TEX's handling of error messages and help texts guarantees that all messages use a uniform structure. Of course, the user must be able to understand contexts like "inserted text" or "to be read again". (See section 2.)

111 error messages fulfill this recommendation.

12. fix with overwrite. Often TEX fixes an error automatically. But the condition that the user must be able to overwrite the fix is sometimes violated. For example, the inserted zero from "Missing number, treated as zero" cannot be overwritten. The recommendation is also fulfilled if either no automatic correction exists or the unchangeable fix is unique.

89 error messages fulfill this recommendation.

13. show additional explanation. As mentioned above all error messages except one have an associated help text. But not all help texts provide an additional explanation. Sometimes it is only a reference, for example, "(See Appendix H.)"; this happens only for error messages that experts might encounter.

108 error messages fulfill this recommendation.

14. customizable amount of explanation. TEX doesn't support this. Users can only decide to request, or not, the help text via 'H' or 'h' at the ?-prompt.

0 error messages fulfill this recommendation.

Summary. Table 1 lists all my decisions; an error message is identified by its line number in `tex.web`.

77.2% of the 14 recommendations are fulfilled by TEX's 111 error messages as I counted fulfillment 1200 times out of a total of 1554. In three recommendations TEX performs poorly: 5, 8, and 14. Without them fulfillment increases to 91.1%. Of course, 5 and 8 especially support beginners.

88.6% of Nielsen's heuristics and 80.4% of the applicable ISO 9241 recommendations are fulfilled.

5 Idiosyncrasies of TEX's error messages

Although the evaluation in section 4 gives not too bad results it is known that TEX behaves in certain situations quite strangely — at least from the viewpoint of a beginner. I select a few topics that seem to irritate some users according to questions and comments on StackExchange.

Cascade of error messages. Not rarely, TEX reports an error, fixes it, and directly displays another error message when the user enters ⟨return⟩. Sometimes it's unavoidable because TEX doesn't allow the user to overwrite its fix. Sometimes it happens because the user doesn't make use of the interactive dialog and fixes TEX's correction.

An example of the first case is

```
! Number too big.
l.2 \dimen0=11111111111
                        111111pt
? h
I can only go up to 2147483647='17777777777="7FFFFFFF,
so I'm using that number instead of yours.

?
! Dimension too large.
l.2 \dimen0=1111111111111111pt

? h
I can't work with sizes bigger than about 19 feet.
Continue and I'll use the largest value I can.
```

TEX replaces the number that is too big by its largest number which is too large for any dimension. (The range for dimensions is smaller than the range for

numbers but the program reuses the *scan_int* procedure when it has to read the numeric part of a dimension [5, §448].) Not only might this behavior irritate a user but also TEX shows in the second error message the user's input although it uses a completely different number.

An example for the second case is (see section 2)

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   \mskip
\,->\mskip
          \thinmuskip
l.4 \,
      $2^3$$
?
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   ^
l.4 \,$2^
         3$$
?
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   \par
l.5
```

The inserted '$' matches with the '$' in front of the '2' so that '^' occurs outside of math mode. As in section 2, TEX inserts another '$'. The first '$' after '3' finishes math mode and leaves another '$' without matching partner. Here is the correct fix:

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>
                   \mskip
\,->\mskip
          \thinmuskip
l.4 \,
      $2^3$$
? 3

\,->\mskip \thinmuskip

l.4 \,
      $2^3$$
? I$
```

That is: Enter "3" to delete the inserted '$', the '\mskip', and the '\thinmuskip'. Now enter "I$" to get '$$' that starts display math mode.

Sometimes it is better just to delete tokens. For example, enter "2" for the input "2^3" to delete TEX's '$' and the '^' so that no math mode is used; with a simple ⟨return⟩ math mode isn't stopped.

Runaway errors. As the name indicates, this type of error is detected by TEX usually much later than when the real error happens. TEX has no chance to find the error that the user made at the place where it occurs. But as TEX implements its own stops it is able to detect the error eventually.

In the following example, the paragraph that starts with "Please press ..." is preceded by an incomplete definition "\def\key#1{{\tt#1}". As the final closing brace is missing TEX adds too much material to the definition. It stops when it finds a control sequence that must not occur in the replacement text of a definition; here it is "\bye".

```
Runaway definition?
#1->{\tt #1} Please press the \key {return} key. \par
! Forbidden control sequence found while scanning
definition of \key.
<inserted text>
                }
<to be read again>
                   \bye
l.11 \bye

? h
I suspect you have forgotten a '}', causing me
to read past where you wanted me to stop.
I'll try to recover; but if the error is serious,
you'd better type 'E' or 'X' now and fix your file.
```

The above example shows a "Runaway definition"; it's one of four runaways. For example, the input "\def\key #1{{\tt#1}} Please press the \key {return key.\bye" with a forgotten brace at the end of \key's argument gives "Runaway argument". TEX reports "\halign{#\bye" as "Runaway preamble" and "\errhelp{Hello-Good\bye" as a "Runaway text".

\outer macros. That \bye is a forbidden control sequence in the above example follows from its definition as an \outer macro [3, p. 206], i.e., the \def is prefixed by \outer. Without the \outer the \bye would appear after the \par in the second line of the above example before TEX displays the runaway error message, and the error message would be "File ended while scanning definition of \key".

\outer macros must not appear in the parameter or replacement text of a macro or in its arguments when the macro is called. Preambles of alignments aren't allowed to contain them, and as seen in the subsection "warnings" of section 2 they cannot be part of a conditional branch that TEX doesn't

execute. With the help of `\outer` macros TEX detects a missing closing brace, a missing `\endgroup` or `\egroup`, or a missing `\fi` much earlier, i.e., only a small part of the document is affected.

Well, nowadays the compilation time of TEX is for most documents quite small and the interactive dialog is not as comfortable as an editor. Thus, people are using a workflow like this: stop the compilation at the first error, fix this error in the input, restart TEX. The early detection of an error through an `\outer` macro to find as many other errors as possible isn't that important any more. So experts sometimes state that the number of problems that `\outer` macros generate is greater than the number of problems that they should simplify.

Nevertheless, I think that certain macros should be defined with the prefix `\outer`. An expert has no problem switching outerness off but, for example, TEX prevents a beginner from putting `\newcount` into a macro that would declare with every call a new register. Another example: The `\outer` macro '`^L`' (ASCII formfeed) can help users to find runaway errors near the place where they are made [3, p. 343].

Alignments. Knuth states [5, §768]: "It's sort of a miracle whenever `\halign` and `\valign` work, because they cut across so many of the control structures of TEX." Well, this also means that alignments can produce a lot of error messages. For example, a missing final brace in an `\halign` throws two error messages: "Missing `}` inserted" and "Missing `\cr` inserted".

Another sort of an error message cascade is seen if the preamble of an alignment contains an error. For example, if a template contains "`\hfi`" instead of "`\hfil`" every use of this template generates the error message "Undefined control sequence". It's not too difficult to find the error as TEX puts the error into the context "template".

Warnings for alignments show one *prototype* row/column, not the whole table [3, p. 302]. TEX always omits the black box from overfull alignments.

Unwanted continuations. I surmise two things: This error surprises beginners and every long-time TEX user encountered this type of error at least once.

When TEX reads a number it continues to read digits until it finds a character that's a non-digit. One space after a number is gobbled by TEX. A beginner might be surprised about what can happen if, for example, a number ends a replacement text.

```
\def\myluckynumber{13}% bad: no space after '13'
My lucky number is \myluckynumber:
\ifnum 13=\myluckynumber 13 is a baker's dozen%
\else it's a nice number\fi. \bye
```

The output is "My lucky number is 13: it's a nice number." When TEX reads `\myluckynumber` in the test of the `\ifnum` it finds next the digits '1' and '3' as the space after the control word is ignored. Thus, TEX executes the test "13=1313" which returns "false". How to fix this? Add a space after "13" in the replacement text of `\myluckynumber`.

A similar situation occurs with TEX's keywords as described in *The TEXbook* [3, p. 298] and mentioned in help texts as "weird error". But sometimes no error is raised as the next example shows.

```
Hello \vskip 0pt plus 1fill Look at this line. \bye
```

The output has two lines; one at the top and one at the bottom of the page. The second line states "ook at this line". As 'l' is a keyword it can be given in lower- or uppercase with spaces in front of it. Thus TEX sees "1fillL" which is a valid ⟨*fil dimen*⟩ [3, p. 271]. (An error occurs with a second line that starts with "Minus" and no dimension.) How to fix this? Add "`\relax`" after incomplete glue specifications and those that end with a ⟨*fil dimen*⟩.

TEX capacity exceeded. This is an example of a fatal error message. The system resources that the system administrator assigned to TEX are too limited to compile the document. *The TEXbook* [3], pages 300–301, explains this error message in more detail and lists the fourteen things that might be mentioned in the error message.

Since its beginning, TEX's parameter settings are well balanced so that large and complex documents can be compiled, for example, *The TEXbook* itself. Modern TEX distributions have increased the limits as modern computers have much more memory than the ones used in the 1980s. Thus it is very unlikely that a document requires more resources except the user has made an error, for example, using `\begingroup` without `\endgroup` in a loop: `\loop \begingroup \ifnum\count0>-1 \repeat`.

A beginner that gets this error message might not be aware that some construction in the document wastes resources. So the beginner is confronted with TEX's hint in the help text "to ask a wizard to enlarge me" (printed in the log file). As modern TEX distributions do an automatic installation on the user's computer the user becomes the system administrator, i.e., the beginner must stand in as the wizard. Of course, TEX cannot help; the beginner must study the documentation of the distribution.

Own error messages. With the pair `\errmessage` and `\errhelp` users can create error messages in TEX's style. TEX provides a default help text as long as `\errmessage` is used without any `\errhelp` given; otherwise the most recent `\errhelp` is shown.

That is, an `\errmessage` without a help text should be preceded by `\errhelp={}`. Unfortunately, TEX's `plain.tex` doesn't follow this advice; see [3, p. 347].

6 TEX's user manual "*The TEXbook*"

What does a beginner learn about error messages from TEX's user manual, *The TEXbook* [3]? A first-time reader, who learns to enter text with at most simple macros, has to read only ≈ 110 pages of [3]. (If one wants to write more complicated macros, seriously typeset mathematics, use `\halign`, etc., the dangerous-bend paragraphs must be read too.) This reader encounters an error message on page 31 in chapter 6 where a compilation run is explained. The message of a commonly-made error is shown: a typo in a TEX command. Last but not least Knuth introduces the interactive dialog with all its features.

Thus a beginner learns a lot about TEX's error handling in an early chapter. However the main chapter about error messages is chapter 27 — the last numbered chapter in [3]. This chapter follows three chapters with summaries of the TEX language. It contains ≈ 3.5 pages for a first-time reader who doesn't stop at the summaries. (Except for a short introduction their 25 pages consist of dangerous-bend paragraphs, i.e., paragraphs that should be skipped during the first reading [3, pp. v and 5].)

Chapter 6 shows a simple error message. Of course, one cannot describe all 111 error messages; the manual is already so thick and complex that it seems to scare off some readers. Thus, a first-time reader who typesets the sentence of exercise 7.1 verbatim cannot be completely prepared for the error messages shown. It might be unavoidable for complex and flexible software like TEX that beginners become frustrated at some point in time.

The commands `\errmessage` and `\errhelp` do not occur in chapters 6 or 27. In fact, `\errhelp` isn't applied in [3]; in Appendix B its use is briefly explained. Chapter 24 contains a short description of `\errmessage` and three appendices show its use.

In chapter 6 readers learn about warning messages for overfull hboxes and read about underfull ones. But it is not easy to find in *The TEXbook* the other warning messages as its index does not contain any entry starting with "warning".

7 Summary and final remarks

The quotation at the beginning of section 3 states that error messages are problematic. Looking at the individual error messages, TEX performs quite well for 10 recommendations out of 14 that were taken from several sources; see section 4. But it fails in two recommendations that especially support beginners. TEX requires that a beginner learns a lot before most error messages are fully understood.

Knuth gave the reporting of error messages in TEX equal treatment to functionality; see section 2. His manual for TEX attempts to prepare readers for error messages in an early chapter; see section 6.

But section 5 shows that TEX's error messages have their own specifics. An experienced user concentrates on the first message and uses the log file to see the whole story. A beginner sometimes tries out stuff that wasn't learned yet but inserted into the document through copy and paste from others. A cascade of unintelligible error messages might be displayed by TEX if this code's usage isn't well documented or studied by the beginner.

Based on the results of section 4 it seems that TEX's error messages might not be the main problem for users. Formats and macro packages must use `\errmessage` and `\errhelp` for their own messages. They should also avoid a user seeing too many contexts from unknown macros in an error message and giving up because of the perceived complexity.

References

[1] Brett A. Becker, Paul Denny, Raymond Pettit, et al., "Compiler error messages considered unhelpful: The landscape of text-based programming error message research", Proceedings of the working group reports on innovation and technology in computer science education (2019), 177–210.

[2] DIN 9241-10:1996, "Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten — Teil 10: Grundsätze der Dialoggestaltung", 1996.

[3] Donald E. Knuth, *The TEXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[4] Donald E. Knuth, "A torture test for TEX", Stanford Computer Science Report *STAN-CS-84-1027*, Stanford, California: Stanford University, 1984.

[5] Donald E. Knuth, *TEX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[6] Donald E. Knuth, *Literate Programming*, CSLI Lecture Note No. 27, 1992.

[7] Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi, "Mind Your Language: On Novices' Interactions with Error Messages", Proceedings of the 10th SIGPLAN Symposium on new ideas, new paradigms, and reflections on programming and software (2011), 3–18.

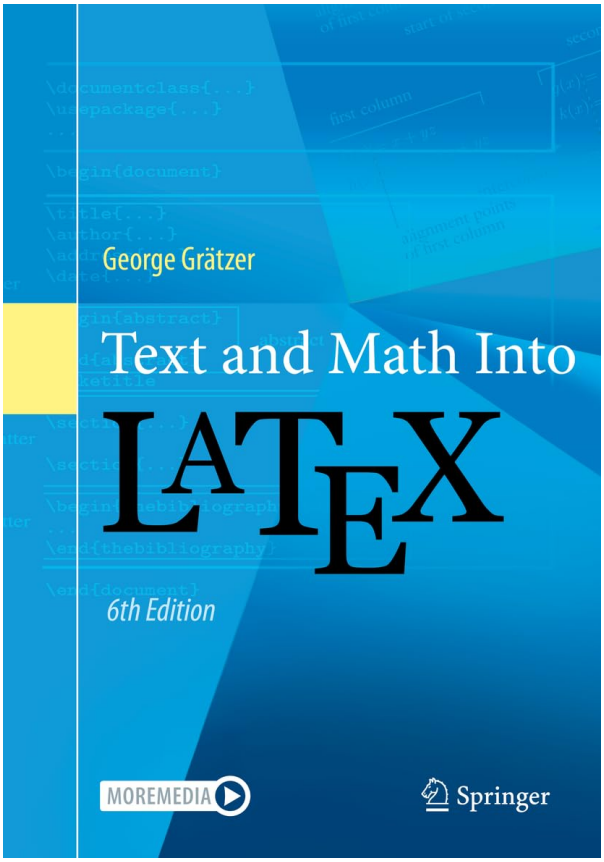
[8] Jakob Nielsen, "Ten usability heuristics" (2005). `en.wikipedia.org/wiki/Heuristic_evaluation`

◇ Udo Wermuth
Dietzenbach, Germany
`u dot wermuth (at) icloud dot com`

Book review: *Text and Math Into LaTeX*, sixth edition, by George Grätzer

Jim Hefferon

George Grätzer, *Text and Math Into LaTeX*, April 2024, 645pp., ISBN 3031552806.
link.springer.com/book/10.1007/978-3-031-55281-6



Text and Math Into LaTeX, 6th edition by George Grätzer is the latest version of a well-known book. Reviewing the prior edition for *TUGboat* was a pleasure so I was happy to also look at this one with the slightly changed title. It is a good update.

1 Overview

The audience is people who write mathematics, including mathematicians, physicists, and computer scientists. The presentation is of consistently high quality: thorough and clear.

As to approach, think of it as a desk reference. The prior edition sits on my desk along with five or six other books that I reach for at least once a week. It reminds me of my options in aligning equations, of how to do a 'see also' in MakeIndex, which of `\makebox`'s optional arguments comes first, and so forth.

It covers, or at least provides a pointer to, the great majority of what an author would need in order to prepare a journal article, a book, or a presentation. And it avoids the error of trying to do everything.

2 Why not the Internet?

Many people look for their LaTeX answers on Stack Exchange[1] or other such sites. These are awesome resources.

However, many LaTeX users lack the vocabulary or mental models to effectively search for an answer or to formulate their own question. I sympathize, since I often find it hard myself. It is a big help to having the material organized so that ideas that are close conceptually are also nearby physically.

Another problem with the Internet is that it is getting old. There is a lot out there that is either outdated or else just wrong. One example is the many pages describing how to switch to entering input in UTF-8, which is now the default. Having a book with updated information is an improvement.

3 Challenge

In approaching this material, there is a tension between being an introduction and being a reference. Mostly Professor Grätzer sticks to being a reference, although there is a brief (about fifty pages) *Short course*. That fits the needs of people who use the tools but are not expert.

The great majority of the book is suitable for dipping into when you have a specific problem. It does not attempt to tell you how to use a text editor, or to explain a whatsit. But a reader who needs a reminder of how to make matrices and matrix variants, along with practical tips, will find that here.

4 Specificity

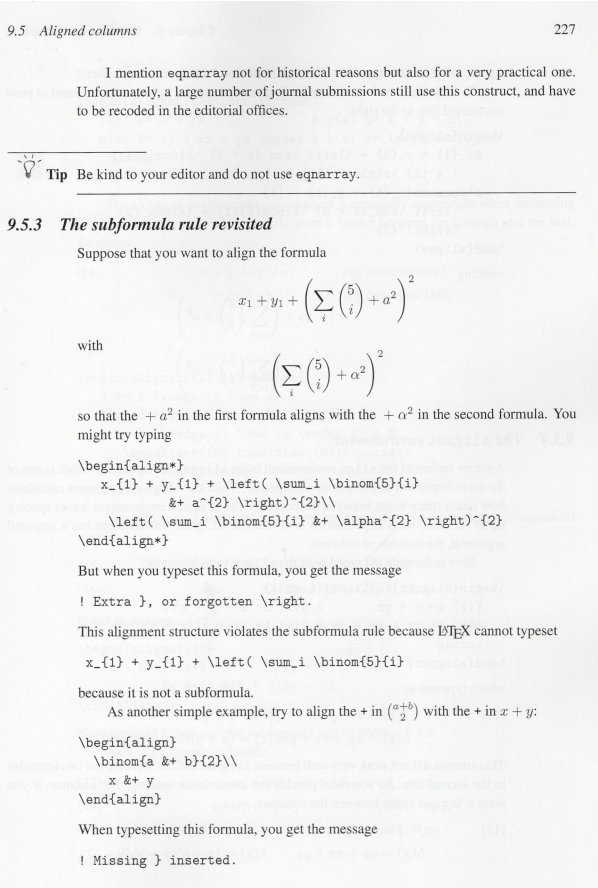
The book is absolutely full of working code. I'll bet that everyone reading this review has found that what helps users see what to do is not syntax diagrams, but code that they can enter.

The book is also full of error messages. When I hear people grouse about LaTeX, always mentioned is puzzling over what a particular output message might mean, and what needs to be changed in the input as a fix. This book is very good at making connections. Not long ago I had a student ask me why `Double subscript` is an error, when it is after all what they were trying to do. Grätzer includes the message along with a contrast between `$a^{b}^{c}$` and `$a^{b^{c}}$`.

[1] `tex.stackexchange.com`

doi.org/10.47397/tb/45-3/tb141reviews-gratzer6

In general, as I mentioned in the prior review, one of the book's strengths is a good sense for what it is that causes users to struggle. The *Tips* particularly shine. Many of them are spot-on for where in my experience people ask about how to accomplish something. This is evident in the Tip in the sample page below.



9.5 *Aligned columns* 227

I mention eqnarray not for historical reasons but also for a very practical one. Unfortunately, a large number of journal submissions still use this construct, and have to be recoded in the editorial offices.

Tip Be kind to your editor and do not use eqnarray.

9.5.3 *The subformula rule revisited*

Suppose that you want to align the formula

$$x_1 + y_1 + \left(\sum_i \binom{5}{i} + a^2\right)^2$$

with

$$\left(\sum_i \binom{5}{i} + \alpha^2\right)^2$$

so that the $+\,a^2$ in the first formula aligns with the $+\,\alpha^2$ in the second formula. You might try typing

```
\begin{align*}
  x_{1} + y_{1} + \left( \sum_i \binom{5}{i}
          &+ a^{2} \right)^{2}\\
   \left( \sum_i \binom{5}{i} &+ \alpha^{2} \right)^{2}
\end{align*}
```

But when you typeset this formula, you get the message

```
! Extra }, or forgotten \right.
```

This alignment structure violates the subformula rule because LaTeX cannot typeset

```
x_{1} + y_{1} + \left( \sum_i \binom{5}{i}
```

because it is not a subformula.

As another simple example, try to align the + in $\binom{a+b}{2}$ with the + in $x + y$:

```
\begin{align}
  \binom{a &+ b}{2}\\
    x &+ y
\end{align}
```

When typesetting this formula, you get the message

```
! Missing } inserted.
```

5 Organizational details

Text and Math Into LaTeX has seven Parts, and seven appendices. First is the short course. Second is *Text into LaTeX*, which is new in this edition and which I will discuss below. Part III is *Fonts for text and math*.

Next is the key Part, *Math into LaTeX*. On my copy of the prior edition this is where the page edges are dirty. It covers many things such as delimiters, symbols, and displayed equations including multiline equations.

Part V is *Document structure*, which discusses the American Mathematical Society article class. In Part VI, there are chapters on hyperlinks, Beamer, and Ti*k*Z (which I will discuss more below). Part VII covers custom commands, including list commands. Finally, Part VIII is *Long documents* and addresses BibTeX, BibLaTeX, and Makeindex.

The appendices have tables for the common math symbols and text symbols, as well as an appendix introducing ChatGPT.

I also noted that very early in the book (page five) is a mention of the TeX Users Group website (`tug.org`). That's great to see, of course.

6 New in this edition

In this edition, broken out as a separate part, is *Text into LaTeX*. It covers eighty pages of things such as paragraphs, making spaces, and making tables. Since one of the strengths of the book is the way that it organizes the material for ease of finding things, I consider this move a good one.

The material on Ti*k*Z has been updated and improved. For drawing I instead use *Asymptote*, so I cannot judge the selection of material here but I'll say that on the forum I go to most often, which is full of beginners, perhaps a quarter of the questions are on this topic. It needs solid coverage. I see that Professor Grätzer thanks Michael Doob for his work on that chapter, so it is good to see that this expanded material has an excellent pedigree.

In the book's final appendix, Professor Grätzer has included an introduction to ChatGPT. Many people use this tool to write LaTeX or Ti*k*Z. A reservation that I have about it is that any introduction will soon be obsolete. However, it is only a few pages and some readers may find it interesting.

7 Omissions

For coverage of most packages, this book defers to the *LaTeX Companion* [1]. That makes perfect sense, although there are some additional packages where perhaps a two or three sentence description and a suggestion to look up more information might be a help. (I'll mention as examples *fancyhdr*[2] and another, perhaps less obvious, *tabularray*.)[3]

8 Conclusion

This is an excellent desk reference, covering a great deal of LaTeX. It is an ideal choice for a school or department library, or as a gift for a working professional or for a graduate student just starting out.

References

[1] F. Mittelbach, U. Fischer. *The LaTeX Companion: Parts I & II*. Addison Wesley, third ed., 2024. `tug.org/l/tlc3`

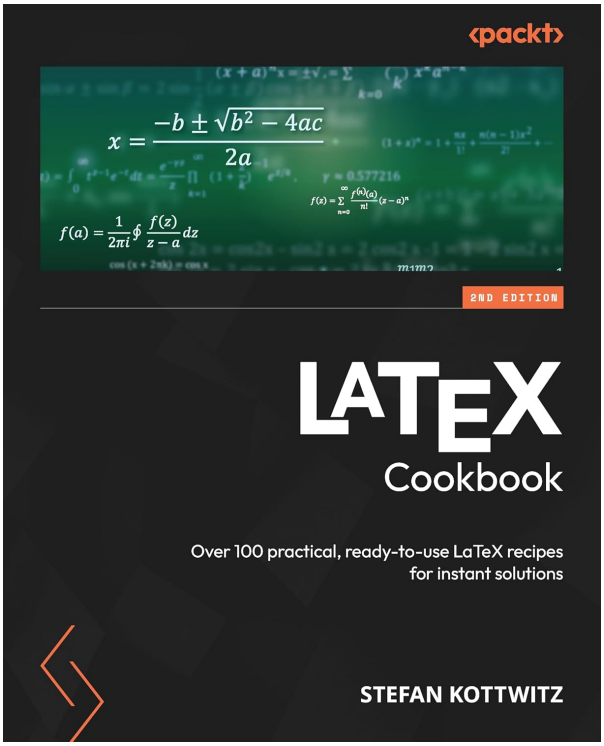
◇ Jim Hefferon
`jim dot hefferon (at) gmail dot com`

[2] `ctan.org/pkg/fancyhdr`
[3] `ctan.org/pkg/tabularray`

Book review: *LaTeX Cookbook*, second edition, by Stefan Kottwitz

Uwe Ziegenhagen

Stefan Kottwitz, *LaTeX Cookbook*, February 2024, 424pp., ISBN 9781835080320. Available in print and electronic formats. `www.packtpub.com/en-us/product/latex-cookbook-9781835080320`



1 The author

Stefan Kottwitz is well known, foremost but not only in the German-speaking TEX world. After his mathematics studies in Jena and Hamburg he has worked as a network and IT security engineer in the maritime industry, equipping cruise ships with network equipment. Besides this full-time job he manages several websites on TEX and LATEX, among them `LaTeX.org`, `goLaTeX.de`, `TeXwelt.de`, and `tikz.net`.

As if this was not enough already, he has written several books on LATEX-centered topics, authoring the *LATEX Beginner's Guide* (in two editions), a book on Ti*k*Z (LATEX Graphics with Ti*k*Z; a Japanese version of it coming in November 2024) and the *LATEX Cookbook* second edition which we will review today. I have known Stefan for more than fifteen years now, so I happily accepted the role of the reviewer.

2 The book

In February 2024 the second edition of Stefan Kottwitz's LATEX Cookbook was published. The second edition of the LATEX Cookbook features 100 LATEX recipes on more than 400 pages, so in comparison with the first edition the amount of text has grown. It is also printed in color while the first edition's images had been printed in grayscale. The book is divided into 13 chapters:

1: Exploring Various Document Classes
The first chapter introduces the basic concepts of LATEX: What "document classes" and "packages" are, and how one can write basic documents such as a small article, thesis or book, or more specialised ones such as a CV, leaflet or poster.

2: Tuning the text
This chapter contains various recipes that deal with the text: its shape, its hyphenation as well as the visualisation of the layout or the typesetting in a grid.

3: Adjusting Fonts
Chapter 3 introduces basic font commands, how to switch a font locally and globally, the output of font tables and the handling of ligatures.

4: Creating Tables
Tables, tables, tables, ... From the design of a table to the positioning to the individual merging of rows and columns, this chapter contains these recipes.

5: Working with Images
This chapter shows how to get the optimal quality, the customization of images, the various shapes and the arrangement of several graphics next to each other.

6: Creating Graphics
This chapter is a small extract of Stefan's book on Ti*k*Z. It does not introduce the general concepts of Ti*k*Z or PGF, but focuses more on practical diagram types: flowcharts, trees, bar and pie charts, etc.

7: Creating Beautiful Designs
While LATEX is mainly used to typeset rather formal standard documents such as articles and books, non-standard "beautiful" designs can be implemented with LATEX as well. This chapter shows how to add background images, create beautiful ornaments and pretty headings, produce a calendar or build a word cloud.

8: Contents, Indexes, and Bibliographies
This chapter deals with more formal aspects of authoring a document. It shows recipes to tweak the table of contents, create and maintain an

doi.org/10.47397/tb/45-3/tb141reviews-kottwitz-cookbook2

index or use LaTeX's magnificent mechanisms for bibliographies.

9: Optimizing PDF Files
Chapter 9 shows how PDF files can be optimized with respect to hyperlinks, metadata or copyright information. It also shows how PDFs can be tweaked for electronic books.

10: Writing Advanced Mathematics
This chapter not only describes advanced math typesetting features but also introduces the main concepts briefly. This part is followed by recipes on the fine-tuning of mathematical content, the plotting of functions in two and three dimensions and mathematical calculations with LaTeX.

11: Using LaTeX in Science & Technology
Chapter 11 is dedicated to the use of LaTeX in science. It covers typesetting algorithms and source codes, the programming with Lua as well as the drawing of molecules, Feynman diagrams, atom representations and several other applications required in MINT sciences.

12: Getting Support on the Internet
In this chapter Stefan explains how to get help for when LaTeX is not producing what the user wants. It introduces various help forms and shows how to produce minimal working examples to get more effective help.

13: Using AI with LaTeX
Since "Artificial Intelligence" is a hot topic, this book needs to have a dedicated chapter as well :-). Stefan shows how ChatGPT can be used to create example LaTeX code, e. g. for bibliographies or more complex math, or to improve one's own LaTeX documents.

3 Conclusion

While I may not be a member of the main target audience of this book any more — that is likely colleague and university students or other people still in the process of learning LaTeX — even I enjoyed reading and reviewing this book.

It shows best practises for a wide range of LaTeX applications and does so quite well. For more experienced LaTeXnicians, this book presents a good overview of how certain requirements can be handled with LaTeX.

◇ Uwe Ziegenhagen
Escher Str. 221
50739 Cologne
Germany
ziegenhagen (at) gmail dot com
https://www.uweziegenhagen.de

Hints & Tricks

Production notes: PDFs and urls

Karl Berry

Documents commonly include PDF, PNG, and JPEG files as images. It's straightforward to extract the latter two from a composed PDF, using, for instance, `pdfimages` from Poppler[1] (or Xpdf). But I wanted to extract an included PDF back out of a document PDF recently, and was surprised to find that there seemed to be no standard tool for this.

I asked Max Chernoff, fellow *TUGboat* TeXnician. He couldn't find anything either, so he wrote a LuaTeX script for the job. It's invoked like this:

```
./luatex-xobject-tub.tex somedoc.pdf
```

The output is written to `luatex-xobject-tub.pdf`, one page for each included PDF. It's available in the *TUGboat* repository.[2]

Another recurring job with PDFs is to check the live urls that are present. (We like to do this before each issue goes to press, so at least we know the urls are working at that time.) Max again came up with a solution, essentially using:

- `qpdf --qdf` (`github.com/qpdf`) for an ASCII transliteration of the PDF;
- `grep --only-matching` to extract the urls, and
- `wget --spider` to check them.

The full script is `check-pdf-urls-tub` in the same *TUGboat* repository directory. The license on these scripts is "do what you want to". Thanks Max!

That first step, converting a compressed PDF into a human-readable form, is generally needed from time to time — i.e., a `pdftype` program analogous to `dvitype` et al. Some other methods I know of:

- Use LaTeX: `\DocumentMetadata{uncompress}`
- Use pdfTeX:
```
\pdfcompresslevel=0 \pdfobjcompresslevel=0
\immediate\pdfximage{in.pdf}%
\pdfrefximage\pdflastximage \end
```
- `pdftk in.pdf output out.pdf uncompress`
- `mutool clean -d in.pdf`

Each has its own benefits and drawbacks, and there are surely yet more out there; I'd appreciate further information.

◇ Karl Berry
github.com/TeXUsersGroup

[1] poppler.freedesktop.org
[2] github.com/TeXUsersGroup/tugboat/blob/trunk/misc/

doi.org/10.47397/tb/45-3/tb141prod

The Treasure Chest

These are the new packages posted to CTAN (`ctan.org`) from April–October 2024. 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*.

◇ Karl Berry
`https://tug.org/TUGboat/Chest`
`https://ctan.org/topic`

fonts

`bonum-otf` in `fonts`
Support for the OpenType font Bonum.

`cascadiamono-otf` in `fonts`
Support for the OpenType font CascadiaMono (with CascadiaCode `fontspec` config files).

`cyrillic-modern` in `fonts/cyrillic`
Cyrillic fonts based on Computer Modern, in Type 1 and OpenType.

`gelasiomath` in `fonts`
Math and small caps for the Gelasio fonts.

`lete-sans-math` in `fonts`
Lato-based OpenType math font for LuaTEX and XETEX.

`luwiantype` in `fonts`
Typesetting package for Hieroglyphic Luwian.

`mfb-oldstyle` in `fonts`
A serif font family for body text, based on Morris Fuller Benton's Century Oldstyle.

`ruscap` in `fonts`
A Metafont for rustic capitals. (See article in *TUGboat* 44:2.)

fonts/utilities

`fontscripts` in `fonts/utilities`
Font encodings, metrics and Lua script fragments for font creation.

graphics

`latexscreenshooter` in `graphics`
Capture screenshots within LATEX documents.

graphics/metapost/contrib/macros

`mp-geom2d` in `graphics/metapost/contrib/macros`
Flat geometry with MetaPost.

`mpkiviat` in `graphics/metapost/contrib/macros`
MetaPost package to draw Kiviat diagrams.

graphics/pgf/contrib

`aiplans` in `graphics/pgf/contrib`
Ti*k*Z-based library for drawing POCL plans.

`polyomino` in `graphics/pgf/contrib`
Polyominoes using Ti*k*Z and LATEX3.

`pgfplotsthemebeamer` in `graphics/pgf/contrib`
Use colours from the current beamer theme in `pgfplots`.

`tikz-decofonts` in `graphics/pgf/contrib`
Simple decoration fonts for short texts.

`tkz-grapheur` in `graphics/pgf/contrib`
Tools for graph plotting and Ti*k*Z.

help

`faq-fr-gutenberg` in `help`
Sources of the GUTenberg French LATEX FAQ available at `faq.gutenberg-asso.fr`.

`wrapstuff-doc-en` in `help`
`wrapstuff` package documentation in English.

info

`asy-overview` in `info`
Overview of the Asymptote language for drawing mathematical graphics.

`wrapstuff-doc-en` in `info`
`wrapstuff` package documentation in English.

macros/latex/contrib

`bib2qr` in `macros/latex/contrib`
Cite BIBTEX entries with QR codes.

`catppuccinpalette` in `macros/latex/contrib`
Provides (x)colors of the catppuccin theme: four pastel color palettes.

`csthm` in `macros/latex/contrib`
Customized theorem environments for computer science documents.

`colorblind` in `macros/latex/contrib`
Easy colorblind-safe typesetting. (See article in *TUGboat* 45:2.)

`commalists-tools` in `macros/latex/contrib`
Operate on comma-separated numeral lists.

`doibanner` in `macros/latex/contrib`
Generate DOI banners and links.

`edmaths` in `macros/latex/contrib`
Report and thesis class for the University of Edinburgh.

`ensps-colorscheme` in `macros/latex/contrib`
Color palette and styling of ENS Paris-Saclay.

`enverb` in `macros/latex/contrib`
Read and/or process an environment verbatim.

`euromoney` in `macros/latex/contrib`
Vector coins and banknotes in euro, with stacking option.

`exercisesheets` in `macros/latex/contrib`
Typeset exercise sheets for classes.

`ezedits` in `macros/latex/contrib`
Tracking document changes and notes.

`fillwith` in `macros/latex/contrib`
Fill vertical space with solid or dotted lines.

`doi.org/10.47397/tb/45-3/tb141chest`

framedsyntax in macros/latex/contrib
Typeset syntax of commands and environments within coloured boxes.
hebdomon in macros/latex/contrib
Generic class for university report writing.
hypcap in macros/latex/contrib
Adjusting anchors of captions (from oberdiek).
interlinear in macros/latex/contrib
Customizable interlinear glossed texts.
keytheorems in macros/latex/contrib
An l3keys interface to amsthm.
linearregression in macros/latex/contrib
Calculate and display linear regressions.
lscapeenhanced in macros/latex/contrib
Extend the lscape and pdflscape packages.
mathgreeks in macros/latex/contrib
Use Greek fonts in math mode.
moremath in macros/latex/contrib
Additional commands for typesetting maths.
numbersets in macros/latex/contrib
Display number sets with customizable typefaces.
passopt in macros/latex/contrib
Passing options to packages or classes.
randintlist in macros/latex/contrib
Random integer number lists, with repeating and sorting options.
rigidnotation in macros/latex/contrib
Typeset vectors and matrices following the RIGID notation.
rpgicons in macros/latex/contrib
Icons for notes on tabletop role-playing games.
rub-kunstgeschichte in macros/latex/contrib
Class for the art history institute at Ruhr University Bochum.
scrhack in macros/latex/contrib
Compatibility package to emulate the former KOMA-Script package.
skillicons in macros/latex/contrib
Integrate skill icons into your documents.
spelatex in macros/latex/contrib
Create PDFs with hyperlinks to audio fragments.
standardsectioning in macros/latex/contrib
Define sectioning commands identical to the standard classes, originally for KOMA-Script.
suanpan-l3 in macros/latex/contrib
Traditional Chinese 7-bids suanpan (abacus) package based on l3draw.
synthslant in macros/latex/contrib
Synthetically slant text.
telprint in macros/latex/contrib
Format German phone numbers (from oberdiek).
tiet-question-paper in macros/latex/contrib
Create exams for TIET, India.
typog in macros/latex/contrib
Typographic fine-tuning and micro-typographic enhancements.
ximera in macros/latex/contrib
Write online interactive content in LaTeX.
xint-regression in macros/latex/contrib
Classical regressions with xint.
xreview in macros/latex/contrib
Reviewing LaTeX documents made easier.
zugferd in macros/latex/contrib
ZUGFeRD and Faktur-X invoicing using LaTeX.

m/l/c/beamer-contrib/themes

beamertheme-edmaths in m/l/c/b-c/themes
Beamer theme for the University of Edinburgh.
beamertheme-gotham in m/l/c/b-c/themes
Versatile theme based on Metropolis.

macros/latex-dev/required

l3backend-dev in macros/latex-dev/required
l3kernel-dev in macros/latex-dev/required
Candidate releases of l3kernel and l3backend, used with latex-dev.

macros/luatex/latex

domaincoloring in macros/luatex/latex
Colored representations of complex functions.
luamml in macros/luatex/latex
Automatically generate MathML from LuaLaTeX math mode material.
semesterplannerlua in macros/luatex/latex
Draw timetables and other organizational matters useful for planning a semester.
tango in macros/luatex/latex
Class for math teachers.

macros/plain/contrib

ifis-macros in macros/plain/contrib
Check if a given input string is a valid number or dimension for TeX. (See articles in *TUGboat* 45:1.)

macros/unicodetex/latex

nxuthesis in macros/unicodetex/latex
Thesis template for Ningxia University.
udiss in macros/unicodetex/latex
LaTeX bundle for typesetting dissertations.

macros/xetex/latex

hduthesis in macros/xetex/latex
Thesis template for Hangzhou Dianzi University.
quran-es in macros/xetex/latex
Spanish translation extension to quran.

Abstracts

Les Cahiers GUTenberg 59, 2024

Les Cahiers GUTenberg is a publication of GUTenberg, the French-language TEX user group (gutenberg-asso.org).

In 2014, the journal PLoS ONE published an article that claims to compare LATEX and Word in terms of efficiency for writing scientific articles. While exhibiting the formal attributes of a scientific article, the text is grossly biased. In addition, it inflicts on readers a lesson in morality, presents a (false) claim of an increased productivity, and concludes with an appeal of denunciation to the authorities that evaluate research.

This article discusses the usefulness of LATEX for lawyers based on the author's own experience. She wrote her PhD in private law using LATEX, which led to the development of a class and BibLATEX styles for references.

Research in history and philology results in long texts according to precise standards. LATEX allows these texts to be highlighted typographically and makes tasks such as bibliography management easier.

This article discusses the updated version of Bernard Gaulle's package `french`. The successive stages of its development are briefly recalled and we emphasise what differentiates it from other methods of writing French.

Currently published on *Amazon*'s *Kindle Direct Publishing* self-publishing platform, *Nénufar* is a collection of literary classics with modernised spelling (1990 rectifications). These books are built by means of LuaLATEX, the *memoir* class, and the *lua-typo* package; in particular, these books aim to emphasise what the TEX family can provide to literary publishing. The overwhelming majority of novels and essays published in French-speaking countries have deplorable typographical quality, which is more related to human incompetence and the limitations of the software currently used than to economic justification.

[Received from Jean-Michel Hufflen.]

doi.org/10.47397/tb/45-3/tb141cahi

La Lettre GUTenberg 53, 2024

La Lettre GUTenberg is a publication of GUTenberg, the French-language TEX user group (gutenberg-asso.org).

Issue #53 published October 8, 2024.
doi.org/10.60028/lettre.vi53

Program for the GUTenberg Day and General Assembly 2024.

Short reports about the monthly online conferences that GUTenberg offers.

April–September 2024: 71 new CTAN packages, including 14 from French-speaking countries.

On the UCA algorithm and index sorting.

This article reviews the composition of paragraphs with Lua, and the use of callbacks.

See `ctan.org/pkg/alegreya`.

Reviews the publication Graphisme en France #30, now also published in English (`cnap.fr/en/publishing/graphisme-en-france-issue-30`);
the high school physics article series in DANTE's *Die TEXnische Komödie*; and French calculation workbooks for high school written in LATEX.

Short news about GUTenberg's desktop wallpapers, DeTi*k*Zify, a follow-up on Ratdolt's bunch of grapes and more.

[Received from Patrick Bideault.]

doi.org/10.47397/tb/45-3/tb141lettre

Die TEXnische Komödie 3/2024

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`dante.de`).

[Received from Uwe Ziegenhagen.]

TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `tug.org/consultants`. If you'd like to be listed, please visit that page.

Boris Veytsman Consulting
132 Warbler Ln.
Brisbane, CA 94005
+1 703-915-2406
Email: `borisv@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 Amnesty International, Annals of Mathematics, ACM, FAO UN, Google, Israel Journal of Mathematics, No Starch Press, Philosophers' Imprint, Res Philosophica, US Army Corps of Engineers, US Treasury, 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.

Dangerous Curve
Email: `khargreaves@gmail.com`

Typesetting for over 40 years, we have experience in production typography, graphic design, font design, and computer science, to name a few things. One DC co-owner co-authored, designed, and illustrated a TEX book (*TEX for the Impatient*).

We can: ▪ convert your documents to LATEX from just about anything ▪ type up your 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. We are a member of TUG, and have supported the GNU Project for decades (including working for them). All quote work is complimentary.

Amy Hendrickson (TEXnology Inc)
100 Centre Street #420
Brookline, MA 02446
+1 617-823-9938
Email: amyh@texnology.com
Web: www.texnology.com

Full time LATEX consultant for more than 30 years; have worked for major publishing companies, leading universities, and scientific journals. Our macro packages are distributed on-line and used by thousands of authors. See our site for many examples: texnology.com.

▪ LATEX Macro Writing: Packages for books, journals, slides, posters, e-publishing and more; Sophisticated documentation for users.

▪ Data Visualization: database publishing.

▪ Innovative uses for LATEX: creative solutions our speciality.

▪ LATEX Training: customized to your needs, on-site or via Zoom. See texnology.com/train.htm for sample of course notes.

▪ Author of newly completed 300+ page e-book *Understanding LATEX: A Visual Guide for Beginners*.

Call or send email, no project too big or too small—I'll be glad to discuss your project with you.

Latchman, David
2005 Eye St. Suite #6
Bakersfield, CA 93301
+1 518-951-8786
Email: contact@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
Boston, MA
Email: enquiries@latextypesetting.com
Web: www.latextypesetting.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 on my web site. 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.

Warde, Jake
90 Resaca Ave.
Box 452
Forest Knolls, CA 94933
+1 650-468-1393
Email: jwarde@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.

Calendar

2024

Oct 17 — The Beatrice Warde Memorial Lecture, St Bride Foundation, London, England. sbf.org.uk/whats-on

Oct 23 – 25 — Grapholinguistics in the 21st century — From graphemes to knowledge, (Donald Knuth is listed as a presenter) Università Ca' Foscari, Venice, Italy. grafematik2024.sciencesconf.org

Oct 31 — Tour of St Bride Foundation, London, England. sbf.org.uk/whats-on

Nov 16 — Journèe GUTenberg 2024, l'École Normale Supèrieure, Paris, France www.gutenberg-asso.fr/Journee-GUTenberg-2024

Dec 3 – 6 — SIGGRAPH Asia 2024, "Curious Minds", Tokyo International Forum, Tokyo, Japan. asia.siggraph.org/2024

2025

Jan 13 – 18 — "Face/Interface", Third international conference on type design, book design and related technologies, Stanford, California, USA. silicon.stanford.edu/face-interface

Mar 1 — **TUG election:** nominations due, 07:00 a.m. PST. tug.org/election

Mar 6 – 8 — TypoDay2025, "Typography and Storytelling", IDC School of Design, Indian Institute of Technology Bombay. Bombay, India. www.typoday.in

Mar 21 — *TUGboat* **46**:1, submission deadline.

Apr 22 – 26 — Association Typographique Internationale, ATypI Copenhagen 2025 Copenhagen, Denmark. atypi.org/conferences-events/atypi-copenhagen-2025

Jun 1 – 6 — Book History Workshop, Texas A & M University, College Station, Texas. library.tamu.edu/book-history

Jun 25 – 27 — Twenty-third International Conference on New Directions in the Humanities, "Oceanic Journeys: Multicultural Approaches in the Humanities", University of Hawaii, Hilo, Hawaii (and online). thehumanities.com/2025-conference

Jul 14 – 18 — Digital Humanities 2025, Alliance of Digital Humanities Organizations, "Building access and accessibility, open science to all citizens", Universidade NOVA de Lisboa, Lisbon, Portugal. dh2025.adho.org

TUG 2025 Trivandrum, Kerala, India

Jul 18 – 20 — The 45th annual meeting of the TEX Users Group. tug.org/tug2025

Jul 7 — *TUGboat* **46**:2, proceedings issue, submission deadline.

Sep 2 – 9 — 25th ACM Symposium on Document Engineering, University of Nottingham, Nottingham, UK. doceng.org/doceng2025

Sep 5 — The Updike Prize for Student Type Design, application deadline, 5:00 p.m. EST. Providence Public Library, Providence, Rhode Island. prov.pub/updikeprize

Status as of 15 October 2024

For additional information on TUG-sponsored events listed here, contact the TUG office by 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. 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.

TUGBOAT Volume 45 (2024), No. 3