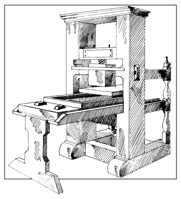


TUGboat

Volume 45, Number 1 / 2024

TEX Users Group

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

Individual memberships

2024 dues for individual members are as follows (all rates will likely have a small increase next year):

- 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 2024 is $125.

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.

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.

About the cover

The cover graphic was created by Federico García De Castro; it illustrates the harmonies in the first prelude of J.S. Bach's *Well-Tempered Clavier*. Federico's article on pages 117–124 describes how it was made, using MetaPost and his MetaGraph macros.

[printing date: April 2024]
Printed in U.S.A.

Addresses

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

Telephone

+1 503 223-9994

Fax

+1 815 301-3568

Web

`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:
`support@tug.org`

Contact the Board of Directors:
`board@tug.org`

A typewriter (or a computer-driven printer of similar quality) that justifies its lines in imitation of typesetting is a presumptuous machine, mimicking the outer form instead of the inner truth of typography.

Robert Bringhurst,
The Elements of Typographic Style
(1992)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
Editor Barbara Beeton

Volume 45, Number 1, 2024
Portland, Oregon, U.S.A.

TUGboat editorial information
This regular issue (Vol. 45, No. 1) is the first issue of the 2024 volume year. The deadline for the second issue in Vol. 45 (the TUG'24 conference proceedings) is July 28, 2024, and for the third (regular) issue, October 6, 2024. The deadline for the first issue of next year is March 23, 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 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, 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`

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.*

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`

From the president

Arthur Rosendahl

What's in a TeX?

The `trip` test defines precisely to what a program must conform for it to be called `tex`, in order to achieve compatibility across operating systems. But the name TeX has for a long been time been used in a less strict way in common parlance, to mean the algorithms of TeX, its "engine", that have been reused in many of its extensions; or, in a yet looser way, the set of programs and tools surrounding TeX, its "ecosystem". That's why we can speak of what "TeX" does as opposed to, for example, Lua in LuaTeX — where many parts of the engine can be rewritten — or as opposed to HarfBuzz in XeTeX and LuaHBTeX.

This polysemy reflects not only the flexibility and adaptability of the algorithms of the original "TeX, the program" — the one that passes the `trip` test — but also, to put it in somewhat immodest terms, its success. It has extended far and wide beyond its originally intended use, gaining in the process not only users but also a sometimes mystical reputation of being the Midas of computer programs, that turns any document into gold. (I would argue that the other part of the Midas legend also applies.) At the same time, the proliferation of extensions of TeX has led to an often confusing choice, starting with the many names that look like "somethingTeX" and that, in the words of a long-time contributor to several of the programs thus named, make some *TUGboat* articles look a bit like a high school magazine. Just have a look at this column if you're not convinced!

This diversity is, however, much more of an asset than a liability, as it gives newcomers a choice — as daunting as that may seem — and enables more advanced users to experiment with different approaches. It is of course a little schizophrenic, though, as was illustrated once at the BachoTeX series of conferences, where in one session entitled "TeX contra TeX" we tried to enact the opposition between the different extensions of TeX as a Western-style duel (as well as a trial; we couldn't quite make up our minds). We didn't issue a judgement, other than it was good that all these different options existed and were somehow united under the banner of "TeX and friends". Very recently, I had the opportunity to discuss the use of TeX in a real court of law; more on that later.

The same ambiguity exists for our organisation, the TeX Users Group, that gathers members from all walks of life, who use TeX and its variants for many different reasons. The prevailing feeling among many long-time members of TUG is that we're more of a *developers'* group, even though we have "users" in our name. This is not necessarily a problem, though, since it is natural that those who are more involved in the organisation become specialists in some area. It is however essential that we continue to attract new users and that they feel welcome. I have never personally had the impression that newcomers were made to feel unwelcome, but this is something to bear in mind.

I was once asked at a conference why there was no LaTeX Users Group and, after being initially startled, outlined some of the above as an explanation of why a user group dedicated specifically to LaTeX would be a bad idea (or perhaps I did actually say my initial thought out loud, namely "what a stupid question" ... belated apologies).[1] As the first TUG president out of the ConTeXt community, I can of course regret that TeX is so often equated with LaTeX, but it is a reality that most users of TeX systems use it through LaTeX (and also, even though they might not be aware of it, pdfTeX).

Whatever road led us to TeX, though, we are all united by a love for typography and beautifully typeset documents, that may take very different shapes and forms. The TeX Users Group has been a place to express this love for over forty years, and it is my hope that it will continue to be such a place for a long time to come. GO FORTH now and create masterpieces of the publishing art!

◇ Arthur Rosendahl
Uppsala University
TUG president
`president (at) tug (dot) org`

[1] Editor's note: For another take on this perennial topic, see the `tug.org/levels` web page: LaTeX vs. MiKTeX.

doi.org/10.47397/tb/45-1/tb139pres

Editorial comments

Barbara Beeton

Changing of the guard: Robin Laakso retiring as TUG Executive Director

For 24 years, Robin has been on her perch in Portland, Oregon, supporting all that TUG is involved in as Executive Director. This has included, but was not limited to, keeping the books, financial and membership records, paying the bills, filing the tax returns, organizing conferences either in cooperation with local organizers or solo (the 2010 conference in San Francisco was a notably successful example), sending materials to and answering questions from members, performing any tasks requested by the board, and overall representing TUG and TeX creditably to the outside world.

Robin attended a number of TUG conferences during her tenure, getting to know members face-to-face. Her last was the 2023 meeting in Bonn, where a photo on page 316 of the proceedings[1] shows her presenting tokens of appreciation to the local organizers (Ulrike and Gert Fischer), as TUG President Boris Veytsman looks on.

Robin has been a good and loyal friend to me as well as a dependable source of information and even a nag when required. All who have known her in this position will miss her, and wish her well in her future pursuits.

Robin's successor is her daughter Sophia, who behind the scenes has assisted Robin with renewals, conference materials, and other tasks, so she is ideally suited to take over the office. Sophia graduated summa cum laude from Oregon State University and has worked for the last several years at the Oregon Historical Society. She will start with the title of Office Manager. We are glad to have Sophia at TUG!

Errata: *TUGboat* 44:2

In my article "What every (LA)TeX newbie should know", the example redefining the command `\i` (shown on page 165) did not have the intended result. Here's another try.

Single-letter commands are also bad candidates for (re)definition by users, as many of them are predefined as accents or forms of letters not usual in English text; `\i` might very well occur with (or without) an accent in a references list. For (a bad) example, consider the author Haïm Brezis:

```
\renewcommand{\i}{\ensuremath{\sqrt{-1}}}
Brezis, Ha\"{\i}m
```
$\Longrightarrow$ Brezis, Ha¨$\sqrt{-1}$m

1 `tug.org/TUGboat/tb44-2/tb137abstracts.pdf`

The example failed because the specified encoding, T1, "normalizes" accented letters, nullifying any attempt to redefine the old commands. This test file will demonstrate what happened.

```
\documentclass{article}
% with T1 enabled, \"\i yields accented i,
% not bad sqrt:
%\usepackage[T1]{fontenc}
\begin{document}
\renewcommand{\i}{\ensuremath{\sqrt{-1}}}
Brezis, Ha\"{\i}m
\end{document}
```

Errata: *TUGboat* 44:3

More than one article in the last issue was corrupted by gremlins, either technical or caused by editorial slipup, or both. We regret the confusion. When possible, corrections have been applied online. Suggestions are given below for possible manual adjustments to the paper version.

- **Janusz Bień, "Towards an inventory of old print characters".** Wrong "old" characters were typeset in several places, one the result of a problem with the default font renderer, and the others for reasons associated with Unicode availability. The detailed explanation is given in a separate erratum by the author, which appears both later in this issue and online in a separate file associated with the original issue.[2]

- **Production notes.** The issue TOC lists "Production notes" on page 449, but opening the paper issue or the complete online issue to that page finds the last page of "The treasure chest", followed immediately by a book review. The intended production notes, which happen not to be specific to the issue, remain online as a separate file[3] and appear on paper in the present issue.

- **George Matthiopoulos, "A short history of Greek type design".** This article (pages 336–353) is illustrated by 61 plates. Unfortunately, owing to a slipup in production, two plate numbers, 8 and 56, are repeated, resulting in erroneous cross references in the text. (The `\label`–`\ref` mechanism wasn't used, eliminating that check.) The two plates with the first instance of the cited plate numbers were packed into `\vbox`es to allow more reliable placement on the page, forgetting that within such a box the value of an automatically stepped counter will be local.

2 `tug.org/TUGboat/tb44-3/bien-rubricella-errata.pdf`

3 `tug.org/TUGboat/tb44-3/tb138prod.pdf`

The problem has been corrected in the online files for the issue, but the erroneous plate numbers are preserved in the printed issue. This can be amended with a manual correction: Starting on page 344, change "plate 8" to "plate 9", and continue, assigning consecutive values, ending with 61. The references will then be correct, since they were hard-coded in the source.

The (effective) end of `comp.text.tex`

The `comp.text.tex` newsgroup has been popularly accessed through Google Groups for many years. But Google has announced that, as of 22 February 2024, they will no longer accept Usenet content in Google Groups. Although Usenet still exists, it becomes ever harder to find active servers (especially ones not overrun with spam). You may want to use one of the many other available methods to seek or provide TeX help, such as those listed at `tug.org/begin.html#help`.

DEK — Puzzles and ChatGPT

A December article in the *New York Times*, "Need a home for 80,000 puzzles? Try an Italian castle",[4] recounts a visit by George and Roxanne Miller (owners of the puzzle-filled castle) to Don Knuth's home in Stanford, California. Don and Mr. Miller had met at a puzzle party years before, and hypothetical puzzles described by Mr. Miller had attracted Don by their mathematical underpinnings. (Algorithms underlying some of these puzzles appear in *TAOCP*.) This led to collaboration on several new puzzles, and a years-long friendship.

Another article[5] contains actual puzzles to be solved, along with (don't look!) the solutions.

Don has also experimented with ChatGPT, describing his experience on his website at Stanford.[6] His questions were simple and straightforward, testing both factual information and ability to emulate various textual styles. As might be expected, where the relevant information had not been included in the training, the "answers" were either evasive or incorrect, or both. But they were expressed in a most literate and often impressive style — likely to be accepted by someone not knowledgeable in the subject area. The potential consequences are terrifying.

Re ChatGPT: this has been my own opinion until recently; however, I've learned that there *are* valid uses for the tool as long as one sticks to areas that are likely to be covered in the tool's training corpus, and carries on a "conversation", fine-tuning the questions to obtain a valid and useful response. I will try to conscript the user who taught me the technique to submit a future article.

[4] `www.nytimes.com/2023/12/29/science/puzzles-mechanical-miller.html`

[5] `www.ageofpuzzles.com/Masters/DonaldEKnuth/DonaldEKnuth.pdf`

[6] `www-cs-faculty.stanford.edu/~knuth/chatGPT20.txt`

An admirable use of AI

The article "AI reads text from ancient Herculaneum scroll for the first time" appeared in *Nature*[7] in October 2023.

The eruption of Vesuvius that buried Pompeii in AD 79 also affected nearby Herculaneum. A library in a villa survived the eruption, but the parchment scrolls held there were carbonized, rendering them unable to be unrolled to read the contents without destroying them. Images were obtained by applying powerful X-ray techniques, which are able to distinguish the ink of the text from the carbonization. A 21-year-old computer science student from the University of Nebraska applied machine learning techniques to detect the first word in an unopened scroll — Greek "porphyras", purple. This breakthrough promises to lead the way to recovering the lost texts of a number of Greek philosophers.

Face/Interface conference at Stanford

On 2 December 2023, a group of font designers, most of them working with languages represented by non-Latin alphabets, gathered for a conference presenting their work in the digital world. With opening and closing keynotes by Chuck Bigelow, the principal in Stanford's short-lived digital typography program (closely associated with the TeX Project), the conference was the kick-off event celebrating Stanford's new SILICON (Stanford Initiative on Language Inclusion and Conservation in Old and New Media) project.[8]

A personal overview of the conference by a member of Stanford's Digital Humanities staff is available online.[9] But I am waiting for an announcement that the actual talks are posted; that has been promised, and is eagerly anticipated.

A longer report on the conference appears in this issue, 7–10.

Accessibility for MathML

"MathML" is short for "Mathematical Markup Language", a member of the family of markup languages intended to direct the formatting of material online and in print.

[7] `www.nature.com/articles/d41586-023-03212-1`

[8] `silicon.stanford.edu`

[9] `digitalhumanities.stanford.edu/face-interface-2023/`

MathML has two distinct components: Presentation MathML (visual layout) and Content MathML, which encodes mathematical semantics without regard for layout and is targeted mainly at computational systems such as Mathematica and Sage. MathML was released as a W3C recommendation in 1998, and standardized by ISO/IEC in 2015. This has been adopted as part of HTML5. Presentation MathML is implemented in major desktop browsers.

The input vocabulary of MathML, unlike that of (original) TeX, is not "native" to a mathematician, although both define how symbols are to be arranged on a surface. The MathML vocabulary, however, is more compatible with that of existing markup languages than is (LA)TeX, and is consequently the preferred form of input for most screen readers; however, ambiguities in possible interpretation need to be resolved, for example, does $|X|$ denote a norm, an absolute value, or something else? That is the direction of the current effort.

The W3C MathML Working Group generally meets weekly by Zoom, and makes the minutes of their meetings available to others interested in the project via an "open" mailing list: `www-math@w3.org`. A subscription can be requested by sending the message `subscription request` to

`www-math-request@w3.org`

It needs to be approved before activation, but acceptance should be without controversy. An archive of the list is held at `lists.w3.org/Archives/Public/www-math/`.

Current activity of the group is concentrated on defining how a MathML-encoded concept can be expressed unambiguously. As an example, what is the best way to voice this expression so that a listener will best understand what is intended:

$$(x, y)$$

marked up as

```
<mrow intent='open-interval($x,$y)'>
  <mo>(</mo>
  <mi arg='x'>x</mi>
  <mo>,</mo>
  <mi arg='y'>y</mi>
  <mo>)</mo>
</mrow>
```

or the same with

```
  intent='ordered-pair($x,$y)'
```

Thanks are due to David Carlisle for clarifying the activities of the working group and providing the example markup.

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

Bibliography of Niklaus Wirth (1934–2024)

Nelson H. F. Beebe

The renowned computer scientist Niklaus Wirth passed away on January 1, 2024. After 26 days of intense work, on January 30 I checked in as version 1.00 this bibliography of his works:

`https://math.utah.edu/pub/bibnet/authors/w/`
`wirth-niklaus.bib, wirth-niklaus.html, ...`

(Further changes will bump the version number to 1.01, 1.02, ...)

The document preamble has a brief resume of Wirth's career, and (not so briefly) discusses the use of Pascal in the rewriting of TeX and METAFONT that led to the 1982 release, and also credits Barry Smith for his work on a polished implementation on the Apple Macintosh. The other languages with which Wirth was involved are also discussed.

My extensive literature searches give me some confidence that I have located almost everything that Niklaus Wirth published in his 58 years of academic activity, but book chapters and technical reports are always hard to find, so a few more might yet surface. I don't have access to a curriculum vitae for him; that may turn up in the future, and permit a cross check of what I have already found in the literature. Contributions welcome!

If we go back to 1980 and ask what programming language Don Knuth could have chosen for the rewrite from SAIL, it seems clear with 44 years of hindsight that Pascal was really the *only choice*, despite its many shortcomings. The bibliography preamble discusses those misfeatures, and gives references to entries later in the file. Brian Kernighan's famous article, "Why Pascal is not my favorite programming language", was worth rereading. I have my own list of gripes about Pascal after using it extensively in the 1980s on DECsystem-20 machines running TOPS-20.

In short, we very likely would not have TeX, METAFONT, and the worldwide TeX community without Niklaus Wirth and Pascal, and that would have made a huge difference in my own professional life. This bibliography is my personal tribute to Niklaus Wirth, with deep thanks for the influence he has had on me. He and I never got to meet in person, and the first view that I ever had of him was earlier this month, watching recent video interviews that are listed in the bibliography. Because of his many years at Stanford and Xerox PARC, I'm sure that he and Don Knuth knew each other well.

◇ Nelson H. F. Beebe
University of Utah

doi.org/10.47397/tb/45-1/tb139beebe-wirth

Face/Interface 2023 conference: Global type design and human-computer interaction

Boris Veytsman

The Face/Interface conference at Stanford (December 1–2, 2023, `face-interface.com`) was not a TeX conference, though it was announced on the TUG home page and in the TUG Mastodon and X feeds. Nevertheless, the connection to TeX was quite prominent: two keynote talks by Chuck Bigelow bookended the conference; DEK attended all presentations; many participants acknowledged that without TeX and LaTeX their work would be much more difficult and tedious; among the keepsakes were offprints of *TUGboat* papers by Jacques André [1] and Kamal Mansour [3], presented as a gift from TUG and Bigelow & Holmes, a teaser for B&H's forthcoming books to be published by TUG, and also a reprint of calligraphy by Kris Holmes. Of course, the topics of the conference, including scripts, typefaces and encodings, also resonate with many members of the TeX community.

This conference marks forty years since the famous ATypI workshop on digital typesetting at Stanford which heralded a new era in typography. Fittingly, the principal organizer of this meeting, Thomas Mullaney, is a Professor of History. Indeed many talks at the conference were devoted to the history of digital typesetting in the last half-century. Thomas himself touched this topic in his opening remarks. Sorin Pintilie discussed the decisions implemented in digital type systems since the 1940s and their evolution. Ferdinand Ulrich presented a rich collection of artifacts from the ATypI workshop and beyond. It is a pity that due to copyright reasons his lecture could not be streamed on YouTube. I was told the recording will be available once stripped of the embargoed materials. I sincerely hope that among the artifacts Ferdinand *could* show are the photos of young DEK, Chuck Bigelow and Hermann Zapf. For now, we can include two photos from the private archive of Prof. Bigelow, cleared for publication in *TUGboat* (Figures 1 and 2).

Of course, the keynote and the final talk by Bigelow also dealt with the history of digital typesetting, with some nostalgic notes about the roads not taken and opportunities missed — for example, the demise of the program of teaching digital typography at Stanford, mostly due to the resistance of certain members of the Stanford arts community.

An interesting contrast with these talks was the presentation by Niteesh Yadav about the challenges for digital typography in the futuristic environment of virtual reality. The lettering there is displayed on an unpredictable background, often badly lit, which requires new solutions and new thoughts. This talk reminded that typography has always evolved answering the challenges of the medium, either ink and paper, or cathode ray tubes, or LCD displays, or low resolution printers — which led to novel technical and aesthetic decisions.

Another look at the way technology influences typography was presented by Shani Avni & Liron Lavi Turkenich, who discussed the changes to Hebrew letterforms brought by the printing press. In

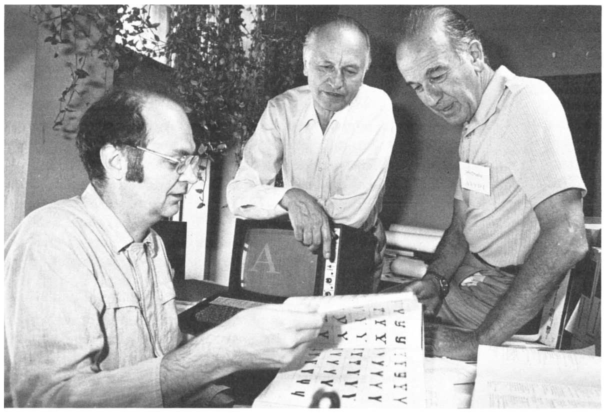


Figure 1: Donald Knuth, Hermann Zapf, and John Dreyfus (former president of ATypI, former typographic adviser to Monotype, book designer, acclaimed type scholar), in Don's office on a hot August day during the 1983 ATypI conference. Photo by an unknown photographer with the Stanford News Department. Reproduced by permission.

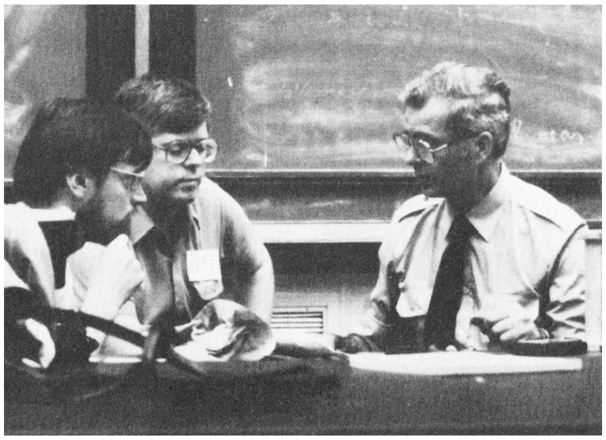


Figure 2: A photo from the 1983 Stanford conference, left to right: organizer Chuck Bigelow (with beard), Raymond Stanley Nelson, Jr. (Smithsonian Museum emeritus scholar of traditional printing technology), Henk Drost (hand punchcutter of the Enschedé type foundry in Haarlem, Holland). Photo credit: Hugh Dubberly. Reproduced by permission.

19th century typesetting, the thin ascenders and descenders tended to break down, and vowel marks were too expensive to add. This led to the considerable changes in the writing system. Some people are now considering undoing these changes, when and as technology allows.

An up-to-the-minute technological challenge is the rapid revolution in machine learning. Arshia Sobhan Sarband discussed the difficulties in training ML models to recognize the complex Arabic script. This talk touched on the topic of contamination of training data for such models: many images of Arabic "in the wild" are Western paintings, where nonsensical signs are used for "oriental exotics".

Another topic of the conference was also related to history: several talks described the creation of digital typefaces for ancient scripts. This is important work: the publication of ancient texts, their indexing and study require a uniform digital representation, both in Unicode and in a faithful typographic rendering. The participants presented a broad range of historical writing systems now being digitized: early Kufi script (Nadine Chahine); ancient coin lettering (Morgane Pierson); oracle bone script in the old China (Zhao Liu & Kushim Jiang); linear Elamite (Sina Fakour) and proto-Elamite (Kaveh Ashourinia); Dives Akuru used in Maldives (Fernando de Moraes Caro); Mayan writing system (Alexandre Bassi & Gabrielle Vail); Egyptian hieroglyphs (Andrey Glass; during the presentation the author announced the release of a new Egyptian font, that severely tests the limits of Unicode and OpenType technology). It is interesting that the Stanford program for digital typography also dealt with historical scripting system and Egyptian typesetting. In Figure 3 we reproduce a part of the PostScript font created by Cleo Huggins in 1988 [2].

One of the most important topics of the conference was the work for very much alive, but underserved languages, those that never had a digital representation before. During decolonization, people often turned to their roots, increasing the interest in their own scripts. It is not coincidental that the talk by Peter Bilak about *Typotheque*, a company involved in the design of fonts for many underserved communities, was titled *Giving Voice to People*. Decolonization involves the "roster of newly empowered voices asking for their narratives to be heard" [5]. One of the sponsors of the conference was SILICON, Stanford Initiative on Language Inclusion and Conservation in Old and New Media (`silicon.stanford.edu`). In his opening remarks Thomas Mullaney, who is also the head of SILICON, talked about its mission. He stressed that it is very important for the specialists outside the user communities not to slip into the role of White Savior, but learn to listen to the voice of the communities themselves, and recognize their right to choose their way of representing their language.

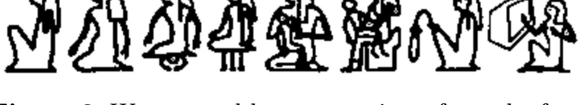


Figure 3: Woman and her occupations, from the font by Cleo Huggins [2]

Besides SILICON, several other organizations working with underserved languages sponsored this meeting or were represented there: the French Atelier National de Recherche Typographique (ANRT, `anrt-nancy.fr`), described by Thomas Huot-Marchand, the Arabic Type Unit at the American University of Beirut (`www.aub.edu.lb/msfea/research/Pages/ATU.aspx`), described by Yara Khoury, and Typotheque from the Netherlands (`www.typotheque.com`), presented by Peter Bilak.

The talks covered many newly digitized scripts, including Balinese (Ariq Syauqi), African languages (Neil Patel; his font specimen book was among the keepsakes of the conference), and Native American scripts digitized by *Typotheque* (Peter Bilak; their font specimen was also among the keepsakes).

In many cases we need not just scripts, but also input methods, as discussed in the talk by Khawar Latif Khan about entering Urdu characters.

Even when a language uses the Latin script, the voice of its own typographers should be heard in developing the fonts. This was a topic of the talk by Thomas Phinney, the recently elected ATypI President, with the strong title, *What if African Designers Created African Latin Fonts?* The author described the Google initiative of commissioning new Latin fonts for African languages from African designers. It is important to note that Google made the principled decision to pay the designers the same rates as used in Europe and North America.

The relation of typography and decolonization remind that the former is a human activity, and thus is closely intertwined with politics and social life. Hrant Papazian mused about this, and wisely noted that the recent move by Kazakhstan from Cyrillic to Latin may mean changing one colonial system to another. Fernando de Moraes Caro in his talk about Dives Akuru noted that after independence the government of Maldives commissioned a book about the script. The talk by Andrew Amstutz about Urdu touched the complex political issues behind the use of Nastaliq for the language. Kourosh Beigpour presented a fascinating journey into the world of Farsi lettering in Los Angeles signs and other inscriptions, where Nastaliq script neighbored Hebrew, Armenian and other letters of the Iranian immigrants. The interplay of social and language aspects was also an important topic of the concluding talk by Chuck Bigelow, who touched on the related topic of language preservation and the difficulty of translation.

The interplay of typography and decolonization is, of course, a huge topic, and a single conference can only scratch the surface. While listening to the talks, I thought about Cyrillic, and how the decolonization of Ukraine was reflected in Ukrainian typography. On the verge of independence Ukraine took the symbolic step of reintroducing the letter Ґ (U+490 and U+491, uppercase and lowercase Cyrillic Ghe with upturn), which had been excluded from the alphabet by Soviet reform. Later attempts to design a contemporary Ukrainian typeface based on both traditions and modernity led to the creation of the Arsenal typeface by Andrij Shevchenko (`github.com/alexeiva/Arsenal`; a LaTeX support package is available at `ctan.org/pkg/arsenal`). The interest in Ukrainian typographic traditions inspired a number of historic typefaces by Bohdan Hdal (`bohdan.com.ua/tvory/t/shryfty`), and Ukrainian Cyrillic forms in the typeface Recht by Andriy Konstantinov (`minttype.com/recht`).

Non-Latin scripts are often very complex. They require the full set of possibilities offered by Unicode and OpenType technology. Therefore it is fitting that several talks at the conference discussed these issues. Neil Patel talked about the challenges of OTF when designing non-Latin scripts. Manish Goregaokar & Ben Yang described the process of adding a new script to Unicode. They used a smart pedagogical device: let's imagine the Latin script is not in Unicode; what hoops do its users need to jump through to get it included? By the way, I was surprised by the fact that Unicode technical committees accept photos of tattoos as examples of script usage.

The talk by Johannes Bergerhausen joined the historical and underserved scripts describing great presentation of Unicode at `decodeunicode.org` and a book with many font samples for different writing systems. Johannes also presented another of his designs: a poster available at `worldswritingsystems.org`. The poster is a smart way to demonstrate the diversity of the writing systems. It takes one character from each script, using different colors for dead and living systems, those represented in Unicode and those not yet there. This poster too was among the conference keepsakes.

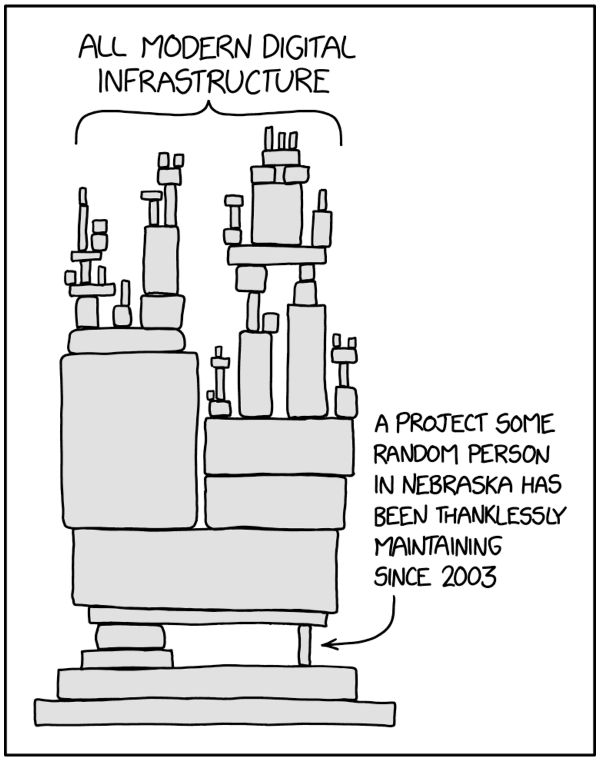

ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

Figure 4: Dependency, from [4]

The huge work required to move a script from the category "not yet digitized" to fully digitized was vividly described in the talk by Anshuman Pandey, who has digitized a large number of scripting systems.

The conference made clear the amount of work done by volunteers and underpaid students. While Unicode and OpenType are now fundamental parts of computing infrastructure, they depend on enthusiasts who are willing to "spend a part of their honeymoon researching Yi syllabary", as one of the presenters at the conference did. I have cited the classic XKCD comic [4] several times in my papers and articles, but cannot help doing it again (Figure 4).

I have mentioned keepsakes several times in this article. The organizers and attendees gave away a large number of beautifully typeset materials (Figure 5), which was all the more surprising for a conference with no registration fee.

To summarize, it was a very interesting conference, showing the deep relationship of typography to history, art, science, technology, and our human way of life.

The organizers promised to post the recordings of the talks, which will be of great interest to the community.

Figure 5: Conference keepsakes

Acknowledgment I am grateful to Barbara Beeton and Chuck Bigelow who kindly shared with me their notes and photos and to Karl Berry for editing the text.

References

[1] J. André. Prehistory of digital fonts. *TUGboat* 44(1):21–57, 2023. doi.org/10.47397/tb/44-1/tb136andre-prehistory

[2] K.C.R. Huggins. Egyptian hieroglyphs for modern printing devices. Master's thesis, Stanford University, June, 1988. apps.dtic.mil/sti/pdfs/ADA326695.pdf

[3] K. Mansour. The non-Latin scripts & typography. *TUGboat* 41(3):275–280, 2020. doi.org/10.47397/tb/41-3/tb129mansour-nonlatin

[4] R.P. Munroe. Dependency, Aug. 2020. xkcd.com/2347/

[5] E.W. Said. *Culture and Imperialism.* Knopf, New York, 1993.

◇ Boris Veytsman
TEX Users Group
borisv (at) lk dot net
https://borisv.lk.net

Typographers' Inn

Peter Flynn

Dashing it off III (em rules reprise)

The inconsistencies I mentioned in *TUGboat* 37:3 about recommendations for dashes have had yet another airing recently.

Conventionally, TEX and LATEX use four horizontal lines in different circumstances:

1. the hyphen (-) is inserted automatically by the hyphenation routine when a word needs breaking at a line-end. Normally, you would only actually type a hyphen when you use a common compound like 'well-founded';
2. the en dash or en rule (–) is primarily used in numeric ranges like 13–22, but sometimes in nonce compounds (see below);
3. the em dash or em rule (—) is used as punctuation — like this — as a form of parenthesis;
4. the minus sign (−) is used only in mathematics.

Different cultures, as well as publishers' house styles, may prescribe other applications, especially for using the en dash in the role of punctuation instead of the em dash, and for putting space before and after the dash—or not, like that. In my summary of the discussion from TYPO-L in *TUGboat* 44:3 (pp. 264–266) I said 'So now we know' — except that we don't.

Such differences appear mainly to arise between the US, UK, and European continental spheres of publishing influence, but probably elsewhere as well (outside my experience, and I'd be interested to hear of others).

The use of the en dash for connecting nonce, rare, or unconventional compounds is interesting, and appears intended to signal their status as once-off, such as 'a Solomon–like judgment', where a hyphen would have implied an established usage.

Breaches of the conventions can reveal strongly-held opinions, as a recent exchange on the BlueSky social network revealed. One example elicited approving comments from journalist and editor Christy Karras and from long-time TEX contributor Don Hosek, confirming the use of the en dash in some cases:

- two words needing to be joined which are not an established hyphenated compound (as in the 'Solomon–like' example above);
- a word needing to join a phrase which is space-separated, like 'New York–based artist';
- a word needing to join a compound already hyphenated, as in the 'Birch–Swinnerton-Dyer conjecture' [2] or 'Smith-Jones–Brown paradox' [1].

Don also felt the general practice is an unspaced en dash for punctuation, giving the example of 'Benjamin Dreyer–style punctuation' but reiterated Cris Maden's point that I mentioned last time, that Tschichold recommended the en dash with spacing, and it is perhaps worth looking at what he said:

> The widely used em dash is a blunt line one em long. This is far too much length and invariably spoils any cultivated type area. The situation could be remedied somewhat by diminishing the word spacing of the line before and after an em dash, but this is easily overlooked.
>
> The only right thing to do is to use lines of half the length, en dashes, and separate them from adjoining words by using the word spacing normal to the line. These en dashes are also called distance lines because they represent the word to in distance or route indications: Basel–Frankfurt; no word spacing is used here. [3, p.149]

Don also noted that the rise of DTP software in the 80s and 90s led to a lot of bad practices becoming common. Christy pointed out that Associated Press (AP) style says to use spaces around em dashes (and doesn't use en dashes at all) and that Chicago style says the opposite: no spaces with an em dash. Don notes the recent update (18th ed.) says to use en-dashes for compounds of two different people's surnames (hyphenated surnames remain unaffected).

The recommendations or requirements of house styles are partly aesthetic, partly practical, and partly connected with their origins in the days of metal type in books, periodicals, and newspapers; and Christy noted that if you are publishing on paper, you still have limits on the amount of space, so formatting may be minimal (in the case of AP, at least).

Ultimately, it usually comes down to following the style of the publisher you are writing or typesetting for. If you are doing it for yourself, or if there are no guidelines or styles, you get a free choice, and I strongly recommend testing different ways with the typeface you are using. The amount of space either side of the en and em dashes may be partly built into the font, possibly making argument redundant about whether or not you should use normal spaces (Tschichold), thin spaces (Don, also *TUGboat*), or none at all. If you *do* use spaces, remember to make the space before the dash a non-breaking space — no-one wants to see a line beginning with a dash.

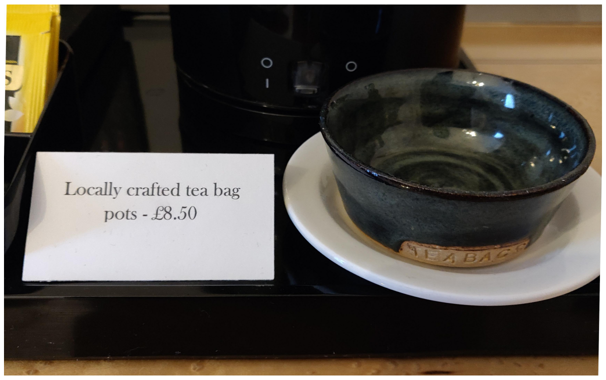
Bookshelves

During the first COVID lockdown, when we were all meeting by group video, I wrote a little document class called bookshelf which turned a BibTeX bibliography into an image of a bookshelf using the title and author fields to fake up the spines with random colors and typefaces.

I am very happy to say that Boris Veytsman has come back to me with a load of suggestions, code, and fixes, so by the time you read this there should be a new version.

Afterthought

I have looked before at examples of the problems raised by poorly-broken centered headings (*TUGboat* issues 33:1, p. 8–10, and 37:3, p. 264–266). Another one cropped up the other day while I was staying at a hotel in England. They very naturally have tea-making equipment in the room, but thoughtfully included a specially-made pot for leaving the used tea-bags in, carefully inscribed 'TEABAGS'. They were available for sale, and the notice is illustrated here. I'll have a locally crafted tea-bag, please. Oh, and a pot.



References

[1] J.A. Barker. A Paradox of Knowing Whether. *Mind*, 84(334):281–283, 1975. `www.jstor.org/stable/2253397`

[2] B.J. Birch, P.H.F. Swinnerton-Dyer. Notes on elliptic curves. II. *Journal für die reine und angewandte Mathematik*, 1965(218):79–108, 1965. `doi.org/10.1515/crll.1965.218.79`

[3] J. Tschichold. *The Form of the Book: Essays on the morality of good design: Dashes.* Lund Humphries, London, 1991.

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

Variable fonts in LuaTEX, with an introduction to the Junicode VF and Elstob fonts

Peter S. Baker

Abstract

This paper introduces variable fonts, now supported by LuaTEX, and explains the benefits this new font technology offers to LuaLATEX users — chief among these being the restoration of some of the typographical capabilities of metal type, nearly lost with the advent of digital fonts. It then describes the capabilities of two variable fonts developed by the author — Junicode VF (where "V" stands for "variable", not "virtual") and Elstob — and briefly introduces the packages that provide access to them, especially the options and commands for controlling their axes.

1 Background

In the days of metal type, long before fonts were scalable, type had to be produced in a variety of sizes. For example, forty-seven pages of the 1798 Caslon *Specimen of Printing Types* [3, pp. 13–105] are devoted to the display of roman and italic type in sizes ranging from "Six Lines Pica" (about 72pt) to "Diamond" (about 4.5pt) — see fig. 1. Type within

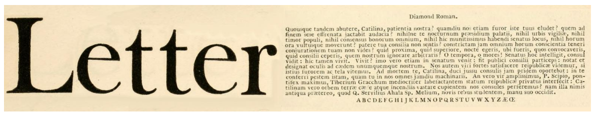


Figure 1: Six Lines Pica and Diamond type, from *A Specimen of Printing Types, by Wm Caslon* [3, pp. [27], [101]].

many font families varied not only in size, but also in shape, with smaller sizes cut proportionally wider, heavier, and with a higher x-height than larger sizes — see fig. 2, where two Caslon specimens, about 12pt and 8pt, have been scaled to the same size. Well into

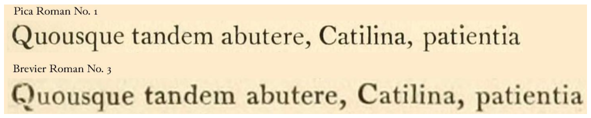

Pica Roman No. 1
Quousque tandem abutere, Catilina, patientia
Brevier Roman No. 3
Quousque tandem abutere, Catilina, patientia

Figure 2: Two sizes of type, from *A Specimen of Printing Types, by Wm Caslon* [3, pp. [77], [97]].

the twentieth century, foundries offered their major typefaces in a wide variety of styles and sizes. For example, the American Type Founders catalogue of 1923 devotes sixty-one pages to a dizzying number of Caslon types — "Caslon Bold Condensed" (6pt to 120pt), "Heavy Caslon" (6pt to 84pt), and "Caslon Openface" (8pt to 48pt), to mention only a few [2, pp. 130–191]. In all cases, smaller and larger types varied in shape as well as in size.

The advent of phototypesetting in the 1950s brought with it the ability to scale type by means of differently powered lenses — but type scaled in this way could vary only in size, not in shape. Although foundries continued to provide a wide variety of styles within type families, typographers increasingly thought scaling alone a good enough solution to the problem posed by the need for differently sized text in printed works. If the type of the footnotes looked anemic compared to that of the body text, that seemed an acceptable price to pay for the convenience and economy of working with a single typeface.

Digital typesetting brought at least the possibility of a return to the practices of early typefounders, and indeed some modern font families feature the kind of variation by size found in the Caslon type catalogue. For example, a number of Adobe "Pro" font families have styles labeled "Caption", "Subhead", and "Display" in addition to the default. Such styles are called **optical sizes** because they are designed to be optically correct within certain size ranges. However, fonts with optical sizes are uncommon and often costly. Most digital fonts belong to so-called RIBBI families, consisting of just four styles — Regular, Italic, Bold, and Bold Italic. Users generally think these styles quite sufficient, and only the most sophisticated typographers bother with such stylistic niceties as those offered by the Adobe "Pro" fonts — especially when they have to be manually selected.[1]

TEX can boast some of the most sophisticated typographical capabilities of any digital publishing system. Donald Knuth's Computer Modern family of fonts (along with several derivatives) has always featured optical sizes similar to those typical of metal type (fig. 3), and TEX automatically selects the cor-

\normalsize
Lorem ipsum dolor sit amet
\footnotesize
Lorem ipsum dolor sit amet

Figure 3: Computer Modern at `\normalsize` and `\footnotesize`, scaled to the same size for comparison.

rect optical size for any run of text set in Computer Modern: users rarely have to think about it.

The introduction of Jonathan Kew's XeTEX in 2004, followed soon afterwards by Will Robertson's fontspec, was a momentous development for TEX.

[1] For a brief history of optical sizes, see Ahrens and Mugikura [1, pp. 17–25].

doi.org/10.47397/tb/45-1/tb139baker-junicodevf

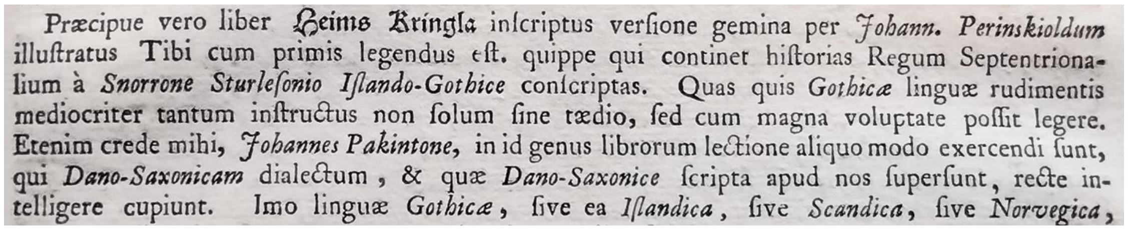

Præcipue vero liber *Heims Kringla* inſcriptus verſione gemina per *Johann. Perinskioldum* illuſtratus Tibi cum primis legendus eſt. quippe qui continet hiſtorias Regum Septentriona-lium à *Snorrone Sturleſonio Iſlando-Gothice* conſcriptas. Quas quis *Gothicæ* linguæ rudimentis mediocriter tantum inſtructus non ſolum ſine tædio, ſed cum magna voluptate poſſit legere. Etenim crede mihi, *Johannes Pakintone*, in id genus librorum lectione aliquo modo exercendi ſunt, qui *Dano-Saxonicam* dialectum, & quæ *Dano-Saxonice* ſcripta apud nos ſuperſunt, recte in-telligere cupiunt. Imo linguæ *Gothicæ*, ſive ea *Iſlandica*, ſive *Scandica*, ſive *Norvegica*,

Figure 4: A sample of Fell's Pica roman and italic type, from Hickes [5].

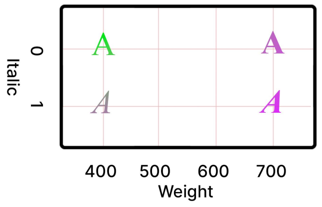


Figure 5: Design space for the four-member Times New Roman font family.

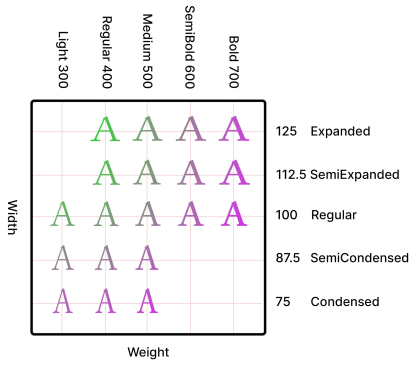


Figure 6: Design space for the nineteen-member Junicode roman subfamily.

Quite suddenly, users gained the ability to access all the fonts installed on their systems, and most of these were in the popular OpenType format.[2] For the typographically ambitious, new vistas opened up.

But with its embrace of OpenType fonts, the TeX community not only benefited from their convenience, but also inherited their limitations. The vast majority of font families installed in any system, and indeed in the CTAN repository, offered only the traditional four styles, while most extended font families — those with more than the four RIBBI styles — were nevertheless stylistically impoverished in comparison with Computer Modern or the immense Caslon family of the 1798 *Specimen*.[3] Twenty years later, font technology has advanced, but we are still struggling to make up the ground we lost in the transition to digital typography.

In font families of the kind I have been discussing, both RIBBI and extended, the collection of styles is organized around one or more **axes** — that is, aspects of a font's design that can change in a systematic way. A font's axes can be pictured as being like the axes in a graph — though in font terminology the graph is called the **design space**. The most common axis is **Weight** — that is, the proportion of black to white in a font's glyphs. Other standard axes are **Width**, **Slant**, and **Optical Size**. **Italic** is also an axis, though one with only two values, since a font either is or is not italic. A RIBBI family positions its four styles at the extremes of two axes, Weight and Italic (see fig. 5), but an extended family may fill in some of the empty spaces in and around the RIBBI design space and add its own axes as well.

The lost ground I mentioned above has been much on my mind as I've worked on the two fonts I am going to discuss in this article. I first developed Junicode in the mid-1990s as a tool for students and scholars of medieval Europe, but it has grown over the decades to support scholars in numerous disciplines (mostly linguistic, literary, and historical). The font is based on types commissioned by John Fell (1625–1686), Bishop of Oxford and a key figure in the early history of the Oxford University Press (see fig. 4).

Fell bequeathed these types to the university, and they were used in many books issued by the Press in the seventeenth and eighteenth centuries. Version 1 of Junicode was a RIBBI family, but in 2019 I began work on a more capable version, released in August 2023. Junicode version 2 is an extended family with three major axes (Italic, Weight, and Width), which are combined in various ways to make thirty-eight styles or **instances** — that is, locations in a font's design space selected and named by the designer. Fig. 6 shows the design space for Junicode's roman face, with its nineteen instances.

[2] See Kew [7] and Robertson [8]. OpenType is a font format, first developed by Microsoft in the 1990s, that enables such features as ligatures, the setting of complex scripts like Arabic and Devangari, and much more.

[3] XeTeX automatically uses the optical sizes of Adobe's "Pro" fonts, but these cannot be included in CTAN — and in any case they offer fewer styles than the larger metal type families.

Junicode is well known to TEX users, having been in the CTAN repository since 2009. With more than 5,000 glyphs, it is an unwieldy thing, and as many of these glyphs have non-standard encodings, it can raise accessibility issues in digital texts of all kinds. Version 2 offers solutions to the accessibility problems of version 1, but the non-standard encodings remain — each of them a potential trap for the unwary user.[4]

With both accessibility and typographical issues in mind, I began work on the Elstob font (`github.com/psb1558/Elstob-font`) in 2018. The first alpha version was released the following year, and the current version is 2.104. Based on another of Bishop Fell's typefaces, Elstob was meant to be lightweight and entirely standards-compliant: while it would not have Junicode's vast character set, its curated selection of glyphs would meet the needs of most medievalists and linguists, and using it would all but guarantee an accessible end product. Further, Elstob would have both Weight and Optical Size axes so that different sizes of type could coexist more comfortably on a page.

Version 1 of Junicode was a static font family — that is, one in which each instance is packaged in its own file. Version 2 also comes in a static version consisting of thirty-eight font files, while the static version of Elstob consists of no fewer than forty-eight files. But each of these fonts has a much more capable and compact variable version as well.

2 Variable fonts

Even extended font families impose significant limitations on users. Because every instance of a family adds a file (in the case of Junicode, a rather large one), many locations in the design space are unavailable to users. Often a large area of the design space may have no instances in it at all. To most users the poverty of the typical static font family will not seem a hardship — after all, it's what we're used to. But what if you could set your chosen typeface in *any* weight and *any* width? If you think you might make use of such an ability, you will like variable fonts.

The specification for variable fonts (the official name is "OpenType Font Variations") first appeared in 2016,[5] and these fonts have been steadily gaining ground ever since. A variable font is one that replaces

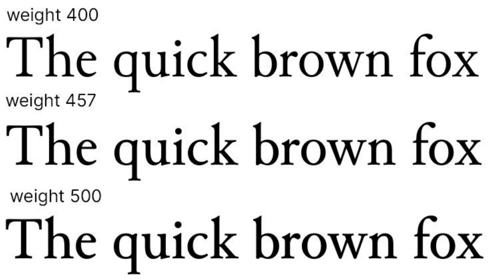


Figure 7: Junicode VF roman at weights of 400 ("Regular"), 457 (custom), and 500 ("Medium").

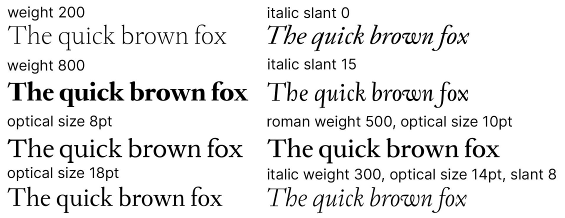


Figure 8: A sampler of styles available with the Elstob font.

the several files of a font face (usually roman or italic) with a single file containing a set of glyph outlines accompanied by deltas governing their transformation. With appropriate software support, outlines can be transformed *continuously* along the font's axes by supplying numerical values. So Junicode VF, the variable version of Junicode, replaces the thirty-eight files of the static font with just two — one for roman and one for italic. Both the static and the variable fonts are based on the same design space, but in the variable font *every possible location* is occupied — not only the blank areas of fig. 6, but also the interstices. If you think the Regular weight of 400 a little too light and the Medium weight of 500 too heavy, you can choose a weight of 420, or 457, or 443.25. See fig. 7, where the difference between 457 and the surrounding weights is subtle, but would make a visible difference in the darkness of a text block.

Like the static Junicode font, Junicode VF is in the CTAN repository, along with documentation and a package for loading the font and accessing its various features. Elstob is not in CTAN, but can be downloaded for free;[6] the latest releases come with a package for TEX users like the one that accompanies Junicode VF.

In addition to Elstob's Weight and Optical Size axes, the italic face includes a Slant axis (see fig. 8).[7]

[4] For more about the accessibility problems raised by Junicode's extensive use of Unicode Private Use Area encodings for specialist medieval glyphs, see the *Junicode Manual* (`github.com/psb1558/Junicode-font/blob/master/docs/JunicodeManual.pdf`), §4.1.

[5] See the OpenType specification [4], especially the section "OpenType Font Variations overview". For an accessible introduction to variable fonts, see Hudson [6].

[6] `github.com/psb1558/Elstob-font`. Like most open source fonts, Elstob can be downloaded from numerous commercial sites, but to obtain the latest version, users should download only from the GitHub repository.

[7] Junicode and Elstob also have specialized axes which are available to TEX users. For Junicode, "Enlarge" lets users set

The current release of Elstob comes with a package like the one for Junicode VF.

3 Support for variable fonts in TeX

Experimental support for variable fonts in luaotfload, the font loader for LuaTeX, first appeared in December 2020.[8] Since that time, support for these fonts has developed rapidly, so that it can now be called mature: if the program has any significant shortcomings or bugs, I have been unable to discover them, though I have been using it almost daily for the better part of a year.

Users should make sure they are running version 3.26 (included in TeX Live 2023) or later of luaotfload, as variable font support is incomplete in earlier versions.

As of version 2.9a (released 2024-02-13), fontspec includes explicit support for variable fonts: for details, see section III.7 of its documentation. Variable fonts can also be managed via fontspec's RawFeature command, present in older versions. fontspec users should always select the HarfBuzz renderer when using variable fonts, as node mode may sometimes fail to load these fonts or apply OpenType features incorrectly (the packages discussed in the next section invoke the HarfBuzz renderer).

XeTeX and other flavors of TeX do not support variable fonts; only LuaTeX.

4 The `junicodevf` and `elstob` packages

The packages for Junicode VF and Elstob are designed to resemble many of the font packages in CTAN — for example, those for `ebgaramond`, `sourceserifpro`, and `roboto` (see Voß [9, p. 299] for a partial list of packages and a link to a complete list). They are loaded in the usual way, with `\usepackage{junicodevf}` or `\usepackage{elstob}`, and they accept a more or less standard set of options, including the following:

extralight (Elstob only) The weight of the main font (that is, the four-style collection selected by fontspec's `\setmainfont` command) is ≈200.

light The weight of the main font is ≈300.

medium The weight of the main font is ≈500.

semibold The weight of the Bold style of the main font is ≈600.

extrabold (Elstob only) The weight of Bold style of the main font is ≈800.

condensed (Junicode VF only) The width of the main font (≈75) is about 85% of Regular (100).

semicondensed (Junicode VF only) The main font is wider than Condensed but narrower than Regular (≈87.5).

expanded (Junicode VF only) The width of the main font is about 115% (≈125) of Regular.

semiexpanded (Junicode VF only) The main font is wider than Regular, but narrower than expanded (≈112.5).

Although these options resemble those for CTAN's static fonts, they produce very different effects, in that they do not produce text in a fixed style, but rather in a range of styles that vary with text size — that is to say, optical sizes, which are supported natively by Elstob and emulated in Junicode by making fine adjustments to the Weight and Width axes (thus the approximation signs in the option list above).

Figs. 9 and 10 illustrate the contrast between Elstob set as body text (11pt) and as footnote text (about 8pt). In fig. 9 the difference in glyph shapes is scarcely visible, but of course that is the point: as early typefounders understood, small type appears to match larger type when the shape is properly adjusted. Fig. 10, which enlarges body text and footnote text to the same size, shows the difference more clearly, the footnote text being heavier and with a higher x-height and shorter descenders than the body text.

For users dissatisfied with the `junicodevf` and `elstob` defaults and the options listed above, two sets of options allow even finer control over these fonts' optical sizing. Here is the first set, which enables adjustments to design choices made via the standard options:

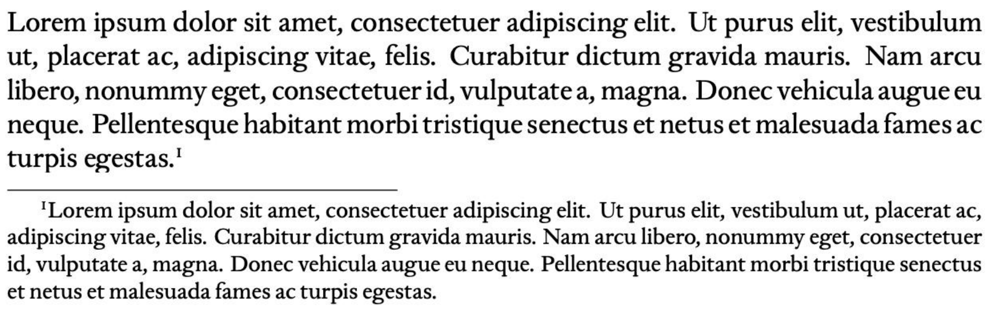
weightadjustment Adjusts the weight of the type by adding this number. For example, if you choose `medium` for the main font (weight ≈500) and `bold` (the default, with weight ≈700), and also include the option `weightadjustment=-25`, then the weights of Medium and Bold text will be lightened by 25 (to ≈475 and ≈675).

widthadjustment (Junicode only) Adjusts the width of the type by adding this number. For example, if you choose `semicondensed` for your document (width ≈87.5), and you also include the option `widthadjustment=5`, then the width will be ≈92.5, between `semicondensed` and `regular`.

opticalsizeadjustment (Elstob only) Adjusts the optical size. By default, the value of this axis is 8 for 8pt text, 12 for 12pt, etc. But if you pass the option `opticalsizeadjustment=-1.5`, the

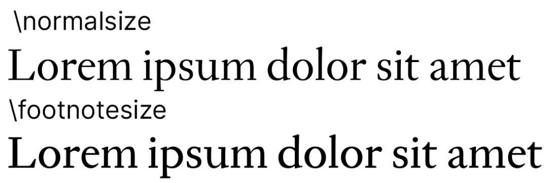
the enlarged lowercase letters that often begin sentences in medieval manuscripts; for Elstob, "Grade" changes the weight of text without changing its width (a more useful feature for web designers than for TeX users), and "Spacing" approximates the word- and sentence-spacing of early metal type.

[8] See the `NEWS` file in the luaotfload repository (`github.com/latex3/luaotfload`, accessed 2024-3-18).


Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.[1]

[1]Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Figure 9: Body text and footnote set in Elstob.


\normalsize

Lorem ipsum dolor sit amet

\footnotesize

Lorem ipsum dolor sit amet

Figure 10: Body text and footnote text enlarged to the same size.

optical size axis will be 6.5 for 8pt type, 10.5 for 12pt, etc. (always staying in the range 6–18).

The second set, to be used instead of the options listed above, gives the user complete control over axis values for every text size in a document. To illustrate, this is the relevant part of the command that loads the `junicodevf` package for the *Junicode Manual*:[9]

```
usepackage[
  MainRegularSizeFeatures={
    {size=8.6,wght=550,wdth=120},
    {size=10.99,wght=475,wdth=115},
    {size=21.59,wght=400,wdth=112.5},
    {size=21.59,wght=351,wdth=100}
  },
  MainItalicSizeFeatures={
    {size=8.6,wght=550,wdth=118},
    {size=10.99,wght=475,wdth=114},
    {size=21.59,wght=450,wdth=111},
    {size=21.59,wght=372,wdth=98}
  },
  MainBoldSizeFeatures={
    {size=8.6,wght=700,wdth=120},
    {size=10.99,wght=700,wdth=115},
    {size=21.59,wght=650,wdth=112.5},
    {size=21.59,wght=600,wdth=100}
  },
  MainBoldItalicSizeFeatures={
    {size=8.6,wght=700,wdth=118},
    {size=10.99,wght=700,wdth=114},
    {size=21.59,wght=650,wdth=111},
    {size=21.59,wght=600,wdth=98}
  },
]{junicodevf}
```

[9] `github.com/psb1558/Junicode-font/blob/master/docs/JunicodeManual.sty`.

For each of the four RIBBI styles, this command defines a list of associative arrays, each prescribing axis coordinates for a range of sizes. In these arrays, a `size` key is mandatory: any array without one is ignored. The arrays should be ordered by point size. The first array prescribes axis coordinates for all sizes up to `size`, the last array for all sizes greater than `size`, and any intermediate arrays a range from the previous to the current `size`. So the ranges covered in each list above are `-8.6pt`, `8.6-10.99pt`, `10.99-21.59pt`, and `21.59pt-`.

Keys other than `size` are the four-letter tags for the font's axes: `wght` (Weight) and `wdth` (Width).[10] When a key is omitted, the default value for that axis is used. When SizeFeatures are given in this way, they override any other options that set or change axis coordinates (e.g. `weightadjustment`).

These lists define only four size ranges because the *Junicode Manual* needs only four; but you can define as many as you need. (The `junicodevf` package, if invoked without options, defines eleven.)

Both the `junicodevf` and `elstob` packages define commands for invoking font styles that match the instances of the corresponding static fonts, plus a few more. These are listed in the fonts' documentation, but as an example, when using Junicode one can switch temporarily from the main font to Condensed Light as follows:

```
{\jCondLight The quick brown fox.}
```

These alternate styles can be customized just as the main styles can. For example, to darken and widen the `\jCondLight` style a little, include this option when loading the `junicodevf` package:

```
CondLightSizeFeatures={
    {size=4,wght=325,wdth=80},
}
```

[10] In OpenType programming, axes are identified by four-letter tags rather than their longer names. By convention, tags for axes defined in the OpenType standard are lowercase, while custom axes are uppercase. The default value of Junicode's ENLA (Enlarge) custom axis is used.

The junicodevf and elstob packages load fontspec (no need to load it again) and depend on that package for all their functionality. The options that manipulate axes generate fontspec commands, while other options are passed through to fontspec. For example, the MainFeatures option allows users to turn on features for all styles of the main font:

```
\usepackage[
    MainFeatures={
        Language=English,
        StylisticSet=9
    }
]{junicodevf}
```

Here, fontspec options set the language of the main font to English and turn on Stylistic Set 9, which modernizes some number-shapes. The same can be done for individual styles of the main font with options like MainBoldItalicFeatures, and for alternate styles with options like CondLightFeatures (features for the Condensed Light style).

Both Junicode and Elstob offer large numbers of OpenType features, and their packages also provide commands for convenient access to a selection of them. For example, Stylistic Set 12 enables transliteration of English text to early English runes for both Junicode and Elstob. One can turn it on either with a fontspec command or with a mnemonic command from the junicodevf or elstob package (with the result shown in fig. 11):

```
fontspec: \addfontfeature{StylisticSet=12}
elstob:   \EarlyEnglishFuthorc
```

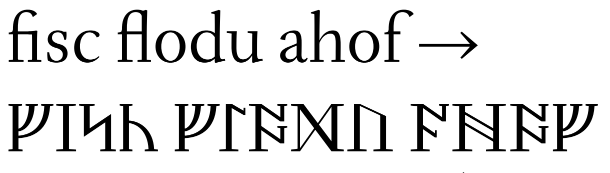


Figure 11: The effect of Stylistic Set 12 (Early English Futhorc) in Elstob.

All of these commands, listed in the Junicode and Elstob documentation, have text variants that work like \textit and \textbf:

```
\textEarlyEnglishFuthorc{fisc flodu ahof}
```

Both Junicode and Elstob, but especially Junicode, also have a number of Character Variant (cvNN) features, which afford access to one or more variants for individual characters. These can be accessed either with fontspec commands or with more compact alternates: (\jcv for Junicode, \ecv for Elstob, \textcv for both). Mnemonics can be used to select from the collection of Character Variant features in either fontspec or junicodevf/elstob commands:

```
fontspec: \addfontfeature{%
              CharacterVariant=\ecvg:1}g
elstob:   \ecv[1]{\ecvg}g
```

Here \ecvg is a mnemonic for 14, identifying Elstob's Character Variant feature for the letter g (cv14). The 1 that appears in both commands is an index that selects the second variant shape of that letter.

5 Conclusion

Junicode VF is, to my knowledge, the first variable font to appear in the CTAN repository, and junicodevf and elstob are first attempts at packages for loading variable fonts in LuaLaTeX. I welcome critiques of these packages, and especially their options for managing the axes of these fonts. As more variable fonts appear in CTAN, it would be useful to standardize the interfaces of any dedicated packages that accompany them.

References

[1] T. Ahrens, S. Mugikura. *Size-Specific Adjustments to Type Designs*. Just Another Foundry, Munich, 2014.

[2] American Type Founders Company. *Specimen Book & Catalogue*. [Jersey City], 1923. archive.org/details/specimenbookcata00amer

[3] W. Caslon. *A Specimen of Printing Types, by Wm Caslon, Letter-Founder to the King*. C. Whittingham, 1798. archive.org/details/specimenofprinti00casl

[4] P. Constable, K. Turetzky, et al. OpenType specification version 1.9, 2022. learn.microsoft.com/en-us/typography/opentype/spec/

[5] G. Hickes. *Linguarum vett. septentrionalium thesaurus grammatico-criticus et archæologicus*. [Oxford University Press], Oxford, 1703-1705.

[6] J. Hudson. Introducing OpenType variable fonts, 2016. medium.com/variable-fonts/https-medium-com-tiro-introducing-opentype-variable-fonts-12ba6cd2369

[7] J. Kew. XeTeX, the Multilingual Lion: TeX meets Unicode and smart font technologies. *TUGboat* 26(2):115–124, 2005. tug.org/TUGboat/tb26-2/kew.pdf

[8] W. Robertson. Advanced font features with XeTeX — the fontspec package. *TUGboat* 26(3):215–223, 2005. tug.org/TUGboat/tb26-3/tb84robertson.pdf

[9] H. Voß. Using OpenType and TrueType fonts with XeLaTeX and LuaLaTeX. *TUGboat* 43(3):295–299, 2022. tug.org/TUGboat/tb43-3/tb135voss-unifont.pdf

◇ Peter S. Baker
b.tarde (at) gmail dot com
https://github.com/psb1558/

dynMath: A PostScript Type 3-based LaTeX package to support extensible mathematical symbols

Abdelouahad Bayar

Abstract

This paper gives an overview of the characteristics and capabilities of a package called dynMath. The main aim of this package is to provide LaTeX with the ability to support dynamic mathematical symbols, thereby improving the quality of the scientific document. Dynamic mathematical symbols are developed in such a way that, when stretched, they will respect *optical scaling*, *uniformity* of shape, *right-sizing* and the closest possible *likeness* to their *analogues* in the *old printing system* (metal typesetting). The tools supplying the stretching with the above characteristics are supported in part by LaTeX (dynMath) and in part by a PostScript Type 3 font.

1 Introduction

In a scientific document, mathematical formulas are written using static and/or variable-sized symbols. In a document typeset at a given size, the shape and the size of static symbols remain unchanged throughout the document. A variable-sized symbol or simply a dynamic symbol varies in terms of size and sometimes shape from one context to another in the same document. The formula in Figure 1, typeset with (normal) LaTeX, is referenced to clarify the concept. The symbols A, a, B, b, C and c are static symbols whereas the parentheses delimiters are dynamic. We notice that the parentheses change in dimension in order to cover the formula to be delimited. In addition, even the shape has changed.

Without a doubt, (LA)TeX is the most widely used system by the scientific community in typesetting scientific documents. (LA)TeX supports the concept of dynamic symbols or operators in all its different implementations, i.e. TeX [8], LaTeX [10], LuaTeX [12], ...

Considering Figure 1, the parentheses delimiting matrix C are not of the same form as those in the case of matrices A and B. So, with (LA)TeX, we lose in *uniformity* of the shape of the parentheses when the height of the formula exceeds a certain level. This is not the case in Figure 2, where the same formulas are reproduced with dynMath.

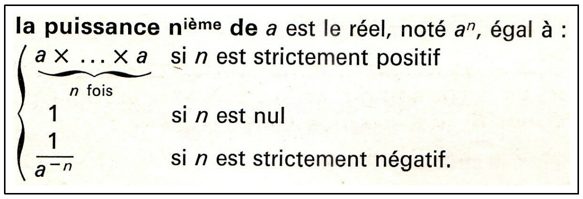
Some of the variable-sized mathematical symbol shapes used in the old printing system (metal typesetting) are supported in digital printing, but only with close shapes, for technical reasons. This causes a slight drop in quality (beauty). The case

$$A = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, B = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, C = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix}$$

Figure 1: Variable-sized parentheses with LaTeX

$$A = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, B = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, C = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix}$$

Figure 2: Variable-sized parentheses with dynMath



la puissance n$^{\text{ième}}$ de a est le réel, noté a^n, égal à :

$$\begin{cases} \underbrace{a \times \ldots \times a}_{n \text{ fois}} & \text{si } n \text{ est strictement positif} \\ 1 & \text{si } n \text{ est nul} \\ \dfrac{1}{a^{-n}} & \text{si } n \text{ est strictement négatif.} \end{cases}$$

Figure 3: Variable-sized braces in metal typesetting

La puissance n$^{\text{ième}}$ de a est le réel, noté a^n, égal à :

$$\begin{cases} \underbrace{a \times \ldots \times a}_{n \text{ fois}} & \text{si } n \text{ est strictement positif} \\ 1 & \text{si } n \text{ est nul} \\ \frac{1}{a^{-n}} & \text{si } n \text{ est strictement négatif} \end{cases}$$

Figure 4: Variable-sized braces with LaTeX

is illustrated by considering an extract from an old mathematics book printed in 1974 [13], as shown in Figure 3. The symbol concerned is the brace. To illustrate the idea, the book's script is reproduced with LaTeX as shown in Figure 4. The property of *resemblance* to analogous metal symbols will be called "*metal-likeness*".

Referring once again to Figure 3, we can see that the two braces are not related by a linear relation. The ratio between heights is not the same as for thicknesses. The scaling model used in document processing (old and current) is not linear. In document processing, this is called *"optical scaling"*. More details on the optical scaling concept are found in [2, 5, 6]. In LaTeX, the scaling used is not linear, though it's also not quite optical, since after a certain size, the thickness becomes constant.

Extensible delimiter symbols in (LA)TeX, when the size of the formula to be delimited exceeds a known level, are obtained by a composition based on more than one character. One of the component characters is repeated as many times as necessary until the size of the delimiter matches, not exactly, but approximately the suggested size. (LA)TEX lacks the property that we'll call "*right-sizing*".

Properties such as uniformity, optical scaling, metal-likeness and right-sizing, which are missing in normal TEX, are principally brought by `dynMath`.

The support of dynamic characters, particularly mathematical symbols, has been a topic of research in the field of electronic document processing for over four decades. It has focused on the extensible Arabic letter to supply the concept of the Kashida, the basis of justification of Arabic scripts [1, 4, 7, 11] especially as it has concerned dynamic mathematical symbols [2, 11].

`CurExt` is a LATEX extension developed to provide the possibility of typesetting mathematical formulas using variable-sized symbols. It supports extensible parentheses and the Kashida, the mechanism needed to extend Arabic mathematical symbols. The `CurExt` package does not offer the ability to manipulate braces and other notations with metal-likeness. Its development principle, taking into account its ability to handle parentheses, enables it, once completed, to meet our four properties of uniformity, optical scaling, metal-likeness and right-sizing.

We should point out that the `dynMath` package is the direct continuation of research published before [3]. The background mathematical model supporting stretching and the TEX macros developed in [3] has been revised and improved to supply as well as possible uniformity, optical scaling, metal-likeness and right-sizing.

The remainder of this paper is organized as follows. In Section 2, we give a brief description of the general layout relative to the `dynMath` package. In section 3, the metal-likeness of dynamic symbols in `dynMath` is defined and discussed. The concepts of "uniformity" and "right-sizing" are studied together in Section 4. In Section 5, the ability to support some old mathematical symbolic notations is presented. In Section 6, the general mode to build the naming of macros to be used by the typesetter is highlighted. Since the paper is a presentation of a LATEX package, some samples in terms of source codes and formatting results are given in Section 7. The paper ends with conclusions and perspectives.

2 General layout of `dynMath`

The `dynMath` package has the ability to support dynamic mathematical symbols due to the existing possibilities of interaction between (LA)TEX and PostScript. This is done using the command `\special` via the `dvips` driver to translate `dvi` files to PostScript [14]. More precisely, we use these methods to include literal PostScript in TEX documents to work with the PostScript Type 3 font. Thus, to format a LATEX source file named `doc.tex` for example, and get the corresponding PDF, the command line needs to be (or the equivalent for LuaLATEX):

```
latex doc.tex; dvips doc.dvi; \
ps2pdf doc.ps
```

At this time, the `dynMath` package consists of two files: `dynMath.sty` and `dynMath.tps`. These two files must be added to the TEX distribution or to the working directory. A brief description of these two files is given below.

- `dynMath.sty`: contains the LATEX package specification in TEX and LATEX programming. It contains all useful variables, macros and interfaces with the content of the `dynMath.tps` file.
- `dynMath.tps`: contains a PostScript Type 3 font named `dynMath` as a literal header that is a '`!`' `\special`. The content of the file looks like : `\special {!...` ⟨*dynMath specification*⟩ `...}`. This is PostScript code inside TEX source, for which we adopted the extension `.tps` to mean TEX (t) and PostScript (ps).

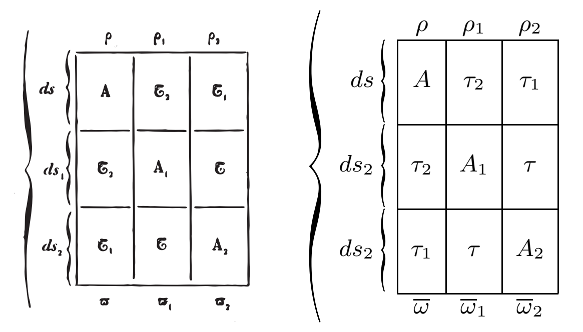
3 Metal-likeness in `dynMath`

The main goal of the research work is the ability to support dynamic characters in document processing. But one of the principal motivations is to add to current document processing tools the capability of supplying stretchable mathematical symbols looking like those printed in metal typesetting. The LATEX package `dynMath` brings this opportunity. In fact, braces supplied by `dynMath` for example (which are among the most difficult symbols to support) look like those from metal typesetting. We call this concept "*metal-likeness*".

An illustration is shown in Figure 5. We consider a mathematical presentation taken from an old book [9] (Figure 5a) and we typeset it using `dynMath` (Figure 5b). There's a difference in the script fonts in the mathematical equation, but this doesn't matter since it's the braces we're interested in.

4 Uniformity and right-sizing

`dynMath` supports dynamic mathematical symbols. It also has the ability to handle these symbols in their exact and tailored sizes or in any size convenient to a context. Let us consider the parenthesis as an example of a dynamic symbol to introduce the concept. We need first to recall in an abstract way how documents are printed through (LA)TEX systems. When printing a document formatted with



(a) Metal left brace symbols

(b) dynMath left brace symbols

Figure 5: Metal and `dynMath` symbols comparison

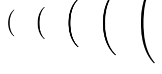


Figure 6: Standalone left parentheses

(LA)TEX, the bitmaps of the characters are used instead of the METAFONT language encoding. It is well-known that applying scaling operations to bitmaps diminishes the quality of the images. Consequently, the support of dynamic fonts based directly on bitmaps is not a feasible way to print documents in good quality.

This problem has already been solved by D. E. Knuth. The approach is presented considering the parenthesis. Knuth has designed standalone parentheses (not composed from other characters) of different sizes as shown in Figure 6. When the height of a mathematical formula is less than the highest of these five parentheses (see Figure 6), the closest parenthesis in terms of size is used to delimit the formula.

The previous idea can not be used to solve entirely the problem since it is necessary to give a large number of parentheses in different sizes to cover all the needs. In addition, we can not predict in a meaningful way a maximum size of mathematical formulas. When a parenthesis, of a height exceeding the highest standalone parenthesis, is needed, (LA)TEX uses a compound parenthesis based on three characters: ⎛, ⎝ and ⎜. The last is repeated as many times as necessary between the first and the second to optimally cover the formula to be delimited. A sample of a compound parenthesis is shown in Figure 7. Two important things to note. The first is that the parenthesis used, whether standalone or compound, will not always be exactly of the same height as the formula to delimit. The second relates particularly to the compound parenthesis: it differs from the first

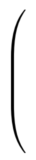
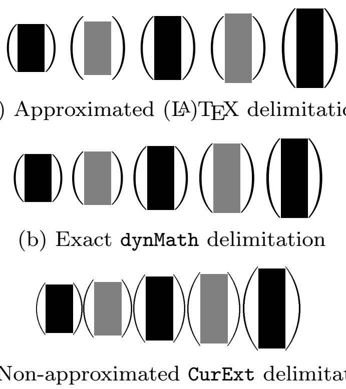


Figure 7: A left compound parenthesis



(a) Approximated (LA)TEX delimitation

(b) Exact `dynMath` delimitation

(c) Non-approximated `CurExt` delimitation

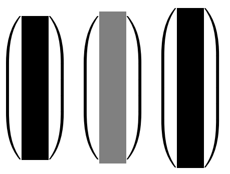
Figure 8: Approximated versus exact delimitations

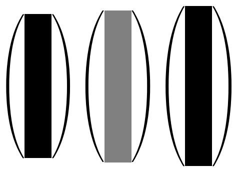
five ones in terms of shape. This leads therefore to a loss of *uniformity*. Let us notice that the symbols in Figure 6 are true parentheses.

Here, we have to point out that `CurExt`, like `dynMath`, also handles the sizes of parentheses in a precise and not approximate way. One difference between `dynMath` and `CurExt` is that in the latter the parentheses' sizes are a little larger than the real size of the formula in order to cover it integrally. This possibility will be supported in the future by `dynMath` as an option.

`dynMath`, developed on the basis of a mathematical model supporting the optical scaling concept, allows for parentheses of sizes satisfying exactly the needs of a formula, using PostScript Type 3 fonts. Even in the case of large formulas, the delimiter produced by `dynMath` retains the form of a parenthesis ensuring the keeping of the "*uniformity*" property. Once again, it must be mentioned that `CurExt` supports these characteristics.

The concept is clarified considering the abstract formulas delimited by parentheses using normal TEX, `dynMath` and `CurExt` in Figures 8a, 8b and 8c. We notice that the abstract mathematical formulas modeled by black boxes in Figure 8a are of sizes equal to existing parentheses in the `cmex10` font. This is why the delimiting parentheses are exactly of the same size (height plus depth). But in the case of gray formulas, only the nearest parenthesis in terms of size is used. With `dynMath`, in all cases the parentheses are exactly of the same size as the formulas to delimit. About `CurExt`, the exact size of the formula is taken and increased a little to determine the


(a) Approximated (LA)TEX delimitation, losing uniformity


(b) Exact `dynMath` delimitation, keeping uniformity

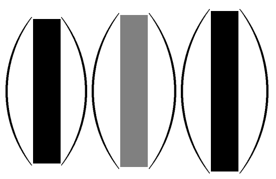

(c) Non-approximated `CurExt` delimitation, keeping uniformity

Figure 9: Approximated delimitation losing uniformity, versus exact delimitation keeping uniformity

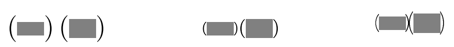

(a) Constant (approximated) (LA)TEX delimitation (b) Exact `dynMath` delimitation (c) Non-approximated `CurExt` delimitation

Figure 10: Approximated versus exact delimitation for global height less than 10pt

size of the parentheses. In this way, it is an exact delimitation or simply a non-approximated one.

Figures 9a, 9b and 9c present the case where the size of a formula exceeds that of the standalone parentheses. Compound parentheses are used with (LA)TEX. As before, (LA)TEX uses approximation in the delimitation process, in contrast to `dynMath` and `CurExt`. In addition, with (LA)TEX, the delimiters lose the real shape of parentheses and so the *uniformity* of delimitation. With `dynMath` and `CurExt`, the general shape is kept.

It's important to mention the case where the overall height (height + depth) of the formula to be delimited is less than 10pt. As shown in Figure 10a, (LA)TEX uses the smallest standalone parenthesis to delimit mathematical formulas in this case. In a way, this is also a case of approximation. On the other hand, `dynMath` and `CurExt` use adapted parentheses (see Figures 10b and 10c).

We can see that the `dynMath` package comes to extend (LA)TEX's capabilities. When processing a document, typesetters can use `dynMath` and the mathematical capabilities of LATEX at the same time

$$n^p = \overbrace{n \times n \times \ldots \times n}^{p\,\text{times}}$$
(a) (LA)TEX over bracing

$$n^p = \overbrace{n \times n \times \ldots \times n}^{p\,\text{times}}$$
(b) `dynMath` over bracing

$$n^p = \underbrace{n \times n \times \ldots \times n}_{p\,\text{times}}$$
(c) (LA)TEX under bracing

$$n^p = \underbrace{n \times n \times \ldots \times n}_{p\,\text{times}}$$
(d) `dynMath` under bracing

$$\dot{\overline{n_1 + n_2 + \ldots + n_p}}$$
(e) (LA)TEX equivalence classes

$$\overset{\cdot}{\overparen{n_1 + n_2 + \ldots + n_p}}$$
(f) `dynMath` equivalence classes

Figure 11: (LA)TEX versus `dynMath` decorations

without any problem. In designing `dynMath`, we insisted on retaining what is common to `dynMath` and normal LATEX, staying faithfully as possible to LATEX, and at the same time adding what can't be supported by LATEX. In Figure 8, we can observe that the thickness, the spaces separating the delimiters and the formulas, in the cases where the size of the formulas coincides with the size of a particular standalone parenthesis, are the same. Otherwise, `dynMath` adds what couldn't be achieved with LATEX in terms of size/shape of parentheses (see Figures 8 and 9).

5 Support of some old mathematical symbolic notations

With advancement in computer technology and software tools, (scientific) document processing has seen great improvements in terms of time, flexibility of processing, quality and so on. But, for technical reasons, many symbols and notations have been replaced by others or completely forsaken. These notations or symbols are the foundation of quality in book typesetting in traditional printing. Parentheses and braces are good examples. With `dynMath`, LATEX can do more of these lost things. We will not show again these two symbols in the delimitation case, but in decoration or diacritics. We consider the LATEX macros `\overbrace` (see Figure 11a), `\underbrace` (see Figure 11c), `\overline` and `\dot` (see Figure 11e) to show more of what `dynMath` can add to LATEX's abilities. The corresponding possibilities supported by `dynMath` are given in Figures 11b, 11d and 11f. With `dynMath`, we can now work in the old way to define equivalence classes, specifying them with dotted parentheses and not just dotted lines (see Figure 11f).

6 Global syntax mode

In `dynMath`, the delimitation of formulas is done with the macro `\meLeft...\meRight`. We followed the command names `\left...\right` but started with a capital letter and preceded by "`me`" meaning metal (referencing the metal symbols). The macros operates in the same way as `\left...\right`, not neglecting the fact that the parameters (formulas) must be enclosed between braces (indicating group) when the formula contains more than one token.

We have followed this way in order to make using `dynMath` especially easy for users that are familiar with typesetting mathematical formulas under (LA)TEX. For example, to handle braces over formulas we programmed the macro `\meOverBrace` of the name coming from the (LA)TEX macro `\overbrace`. We will follow this method when implementing what remains to complete the package.

7 Samples

In this section, we will present some samples of formulas formatted with `dynMath`, also giving the corresponding LATEX. In particular, the code of some examples used in the previous section will be considered in the current one.

`dynMath` requires the package `mathstyle` and so the latter must be installed in the (LA)TEX distribution used. We recall that the PostScript Type 3 font used as the base of `dynMath` is provided in the file `dynMath.tps`. Thus, both `dynMath.sty` and `dynMath.tps` also have to be added to the (LA)TEX distribution or copied into the directory containing the LATEX file to format.

The document to format using `dynMath` must contain the usual line `\usepackage{dynMath}`.

We will give some samples of LATEX sources based on `dynMath` followed by the corresponding generated output. To compare with normal LATEX, we use the same source code with `\left` and `\right` instead of `\meLeft` and `\meRight` respectively.

7.1 Sample one: parentheses

`dynMath` source:

```
\[ \meLeft({
\begin{array}{cc}
\meLeft({
\begin{array}{cc}
a_{11}& a_{12}\\a_{21}& a_{22}\\a_{31}& a_{32}
\end{array}
}\meRight)
& ... bij components  \\
  ... cij components & ... dij components
}\meRight)
\end{array}
}\meRight) \]
```

`dynMath` corresponding output:

$$\left(\begin{array}{cc}\left(\begin{array}{cc}a_{11}&a_{12}\\a_{21}&a_{22}\\a_{31}&a_{32}\end{array}\right)&\left(\begin{array}{cc}b_{11}&b_{12}\\b_{21}&b_{22}\\b_{31}&b_{32}\end{array}\right)\\\left(\begin{array}{cc}c_{11}&c_{12}\\c_{21}&c_{22}\\c_{31}&c_{32}\end{array}\right)&\left(\begin{array}{cc}d_{11}&d_{12}\\d_{21}&d_{22}\\d_{31}&d_{32}\end{array}\right)\end{array}\right)$$

LATEX source:

```
\[ \left({
\begin{array}{cc}
...
\end{array}
}\right) \]
```

LATEX corresponding output:

$$\left(\begin{array}{cc}\left(\begin{array}{cc}a_{11}&a_{12}\\a_{21}&a_{22}\\a_{31}&a_{32}\end{array}\right)&\left(\begin{array}{cc}b_{11}&b_{12}\\b_{21}&b_{22}\\b_{31}&b_{32}\end{array}\right)\\\left(\begin{array}{cc}c_{11}&c_{12}\\c_{21}&c_{22}\\c_{31}&c_{32}\end{array}\right)&\left(\begin{array}{cc}d_{11}&d_{12}\\d_{21}&d_{22}\\d_{31}&d_{32}\end{array}\right)\end{array}\right)$$

7.2 Sample two: braces

`dynMath` source:

```
\[
\meLeft\{{
 \begin{array}{l}
  \meLeft\{{
   \begin{array}{lcl}
    f_{1}\meLeft({x_{1},x_{2},\cdots,x_{9}}
           \meRight) & = & F_{1}\\
    f_{2}\meLeft({x_{1},x_{2},\cdots,x_{9}}
           \meRight) & = & F_{2}\\
    f_{3}\meLeft({x_{1},x_{2},\cdots,x_{9}}
           \meRight) & = & F_{3}
   \end{array}
  }\meRight.\\
  \\
  \meLeft\{{
   \begin{array}{lcl}
    ... gi equations
   \end{array}
  }\meRight.\\
  \\
  \meLeft\{{
   \begin{array}{lcl}
    ... hi equations
   \end{array}
  }\meRight.
 \end{array}
}\meRight.
\]
```

dynMath corresponding output:

$$\left\{ \begin{array}{l} \left\{ \begin{array}{lcl} f_1(x_1,x_2,\cdots,x_9) & = & F_1 \\ f_2(x_1,x_2,\cdots,x_9) & = & F_2 \\ f_3(x_1,x_2,\cdots,x_9) & = & F_3 \end{array} \right. \\ \\ \left\{ \begin{array}{lcl} g_1(x_1,x_2,\cdots,x_9) & = & G_1 \\ g_2(x_1,x_2,\cdots,x_9) & = & G_2 \\ g_3(x_1,x_2,\cdots,x_9) & = & G_3 \end{array} \right. \\ \\ \left\{ \begin{array}{lcl} h_1(x_1,x_2,\cdots,x_9) & = & H_1 \\ h_2(x_1,x_2,\cdots,x_9) & = & H_2 \\ h_3(x_1,x_2,\cdots,x_9) & = & H_3 \end{array} \right. \end{array} \right.$$

LaTeX source:

```
\[
\left\{{ \begin{array}{l}
...
 \end{array}
}\right. \]
```

LaTeX corresponding output:

$$\left\{ \begin{array}{l} \left\{ \begin{array}{lcl} f_1\left(x_1,x_2,\cdots,x_9\right) & = & F_1 \\ f_2\left(x_1,x_2,\cdots,x_9\right) & = & F_2 \\ f_3\left(x_1,x_2,\cdots,x_9\right) & = & F_3 \end{array} \right. \\ \\ \left\{ \begin{array}{lcl} g_1\left(x_1,x_2,\cdots,x_9\right) & = & G_1 \\ g_2\left(x_1,x_2,\cdots,x_9\right) & = & G_2 \\ g_3\left(x_1,x_2,\cdots,x_9\right) & = & G_3 \end{array} \right. \\ \\ \left\{ \begin{array}{lcl} h_1\left(x_1,x_2,\cdots,x_9\right) & = & H_1 \\ h_2\left(x_1,x_2,\cdots,x_9\right) & = & H_2 \\ h_3\left(x_1,x_2,\cdots,x_9\right) & = & H_3 \end{array} \right. \end{array} \right.$$

7.3 Sample three: Figure 3

The mathematics in Figure 3 was typeset in native LaTeX in Figure 4. It remains to typeset it with dynMath, as follows. The dynMath source:

```
\begin{minipage}{7cm}
\textbf{La puissance n$^{\mbox{\scriptsize
  i\`{e}me}}$ de }$a$ est le réel,
  noté $a^{n}$,\\ égal~à~:\\
$
\meLeft\{%
  \begin{array}{ll}%
    \meUnderBrace{a\times\ldots\times a}
      _{n\textrm{ fois}} &
    \kern -3pt\textrm{si }n
      \textrm{ est strictement positif}\\
    1 & \kern -3pt\textrm{si }n
      \textrm{ est nul}\\
    \frac{1}{a^{-n}} & \kern -3pt\textrm{si }n
      \textrm{ est strictement négatif}
  \end{array}
\meRight.
$
\end{minipage}
```

dynMath corresponding output:

La puissance n$^{\text{ième}}$ de a est le réel, noté a^n,
égal à :

$$\left\{ \begin{array}{ll} \underbrace{a \times \ldots \times a}_{n\text{ fois}} & \text{si } n \text{ est strictement positif} \\ 1 & \text{si } n \text{ est nul} \\ \frac{1}{a^{-n}} & \text{si } n \text{ est strictement négatif} \end{array} \right.$$

7.4 Sample four: over brace

dynMath source:

```
\[ n^p=\meOverBrace{n\times n\times\ldots
    \times n}^{p\, \mathrm{times}} \]
```

dynMath corresponding output:

$$n^p = \overbrace{n \times n \times \ldots \times n}^{p\,\mathrm{times}}$$

LaTeX source:

```
\[ n^p=\overbrace{...} \]
```

LaTeX corresponding output:

$$n^p = \overbrace{n \times n \times \ldots \times n}^{p\,\mathrm{times}}$$

7.5 Sample five: under brace

dynMath source:

```
\[ n^p=\meUnderBrace{n\times n\times\ldots
    \times n}_{p\, \mathrm{times}} \]
```

dynMath corresponding output:

$$n^p = \underbrace{n \times n \times \ldots \times n}_{p\,\mathrm{times}}$$

LaTeX source:

```
\[ n^p=\underbrace{...} \]
```

LaTeX corresponding output:

$$n^p = \underbrace{n \times n \times \ldots \times n}_{p\,\mathrm{times}}$$

7.6 Sample six: over dot

dynMath **source:**

```
\[ \dot{\meOverParenthesis{n_{1}+n_{2}+
     \ldots + n_{p}}} \]
```

dynMath corresponding output:

$$\dot{\overparen{n_1 + n_2 + \ldots + n_p}}$$

LaTeX source:

```
\[ \dot{\overline{...}} \]
```

LaTeX corresponding output:

$$\dot{\overline{n_1 + n_2 + \ldots + n_p}}$$

So far, we've talked in terms of concepts and examples about parentheses and braces as dynamic symbols. The latter represent the category of symbols with extensible parts based on curvilinear (nonlinear) curves. They remain among the most difficult to support in `dynMath`. This is not the case for extensible symbols whose dynamic parts are purely lines, which are easy to parameterize for extension. Square brackets are a good example. The following sample shows the delimitation of a multi-matrix by square brackets.

7.7 Sample seven: square brackets

`dynMath`:

$$\begin{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} & \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix} & \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \\ d_{31} & d_{32} \end{bmatrix} \end{bmatrix}$$

LaTeX:

$$\begin{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} & \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix} & \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \\ d_{31} & d_{32} \end{bmatrix} \end{bmatrix}$$

The source for the above formulas is the same as in Sample 1, except that the third lines of the two high matrices are removed and the delimiters '(' and ')' are replaced by '[' and ']' respectively.

8 Conclusions

`dynMath` is a LaTeX package that supports dynamic mathematical symbols with respect to *optical scaling*, *metal-likeness*, *uniformity* and *right-sizing*. It will be finalized soon and deposited in the CTAN (Comprehensive TeX Archive Network) repository with a complete user manual. As the symbols are programmed in PostScript, they are not affected by the color effects controlled by LaTeX. As a future enhancement, `dynMath` will be extended to support color interaction with PostScript.

References

[1] A. Anane. Arabic text justification using LuaLaTeX and the DigitalKhatt OpenType variable font. *TUGboat* 42(3):247–257, 2021. `https://tug.org/TUGboat/tb42-3/tb132anane-variable.pdf`

[2] J. André, I. Vatton. Dynamic optical scaling and variable-sized characters. *Electronic Publishing*, 7(4):231–250, 1994. `https://jacques-andre.fr/japublis/opticalscaling.pdf`

[3] A. Bayar. Towards an operational (La)TeX package supporting optical scaling of dynamic mathematical symbols. *TUGboat* 37(2):171–179, 2016. TUG 2016 (Toronto) conference proceedings. `https://tug.org/TUGboat/tb37-2/tb116bayar.pdf`

[4] D.M. Berry. Stretching letter and slanted-baseline formatting for Arabic, Hebrew, and Persian with ditroff/ffortid and dynamic PostScript fonts. *Software: Practice and Experience*, 29(15):1417–1457, 1999. `https://cs.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/keshide.paper.pdf`

[5] Circuitous Root. Clubs and cults revisiting the concept of 'typeface' and the optical scale in typefounding, 2013. `https://www.circuitousroot.com/artifice/letters/press/typemaking/making-matrices/terms/logical-grouping/clubs-and-cults/index.html`

[6] Circuitous Root. From the optical scale to optical scaling, 2013. `https://www.circuitousroot.com/artifice/letters/press/typemaking/mats/optical/index.html`

[7] M. Elyaakoubi, A. Lazrek. Justify just or just justify. *Journal of Electronic Publishing*, 13(1), 2010.

[8] D.E. Knuth. *The TeXbook*. Addison-Wesley, Reading, Massachusetts, 1st ed., 1984.

[9] G. Lamé. *Leçons sur les coordonnées curvilignes et leurs diverses applications*. Imprimerie de Mallet Bachelier, Rue du Jardinet 12, Paris, 1859. `https://archive.org/details/leonssurlescoor01lamgoog`

[10] L. Lamport. *LaTeX — A Document Preparation System*. Addison Wesley, USA, 1994.

[11] A. Lazrek. CurExt, typesetting variable-sized curved symbols. *TUGboat* 24(3):323–327, 2003. EuroTeX 2003 (Brest) conference proceedings. `https://tug.org/TUGboat/tb24-3/lazrek.pdf`

[12] LuaTeX development team. *LuaTeX Reference Manual*. `https://ctan.org/pkg/luatex`

[13] M. Monge, M.C. Audouin-Egoroff, F. Lemaire-Body. *Arithmetiques, Analyse et Probabilités, Terminal C et E*. Librairie Classique Eugène Belin, France, 1974.

[14] T. Rokicki. *Dvips: A DVI-to-PostScript Translator*. `https://tug.org/dvips`

◇ Abdelouahad Bayar
Cadi Ayyad University — Higher School of Technology of Safi
Sidi Aissa Road, PB 89
Safi, 46000
Morocco
`a.bayar (at) uca dot ma`
ORCID 0000-0002-3496-505X

Tracing bitmap fonts in LMTX

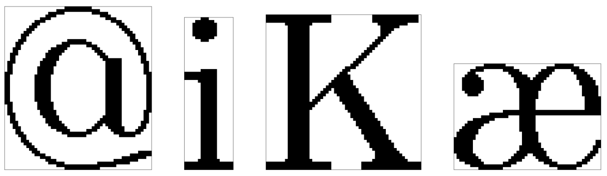
Hans Hagen, Mikael P. Sundqvist

This is a follow up on potrace-generated outlines from bitmaps (see preceding article, “Unusual bitmaps”). Most of today’s TeX users have probably never generated a document with bitmap fonts but back in the day we only had these, almost always Computer Modern. Because we don’t use bitmap fonts in ConTeXt there is no need to support so-called PK (bitmap) fonts in the backend, but, because we did support them in MkIV, we still have it in LMTX. After all, what is TeX without bitmap fonts? We also owe it to Don Knuth.

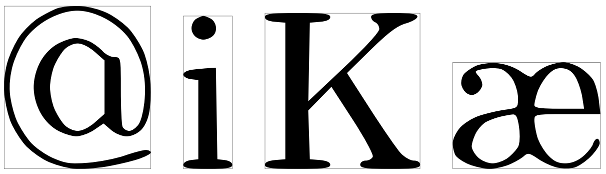
It is a bit of challenge to load a traditional eight bit font using only a TFM file because in ConTeXt we map everything to Unicode. Regular Type 1 fonts are still supported, although they are declared obsolete, and vendors have moved on to OpenType fonts. When we define a font that has Type 1 resources, we load it as if it were an OpenType font. These eight bit fonts thus become wide (up to 64K glyphs) fonts internally. Normally we consult the AFM and PFB files and leave the TFM files, if present at all, for what they are. You can think of runtime `afmtotfm` conversion. An exception is the few traditional math fonts that we support: Antykwa, Iwona and Kurier; here the TFM files provide dimensions.

For the purpose of demonstrating what comes next it is enough to know that in principle one can still mess around with TFM values, either outline or bitmap, and that encoding files play a role in mapping them onto Unicode. We never set out to do something like this, but, because we had the loaders available anyway, some quick (few line) experiments of passing PK bitmaps to the MetaPost potrace helpers we got curious.

If you’ve run into an old TeX document on the web you might have noticed bitmap fonts being used. Often these are of a relatively low resolution. The reason that one never noticed that in print is that, when read on screen, we see glyphs large and can even zoom in, while in print they are seen small. When larger glyphs are used (say in a title) the scaled glyphs are actually different bitmaps: they have the same resolution as the smaller ones but more pixels because they are larger. Take these characters at 600 dpi, scaled up from their original 10 point size:



How do these patterns translate into an outline? For this we use the potrace library that we have available in LuaMetaTeX and interfaced to Metafun in ConTeXt (as discussed in the companion article), although for fonts we follow a more direct route.



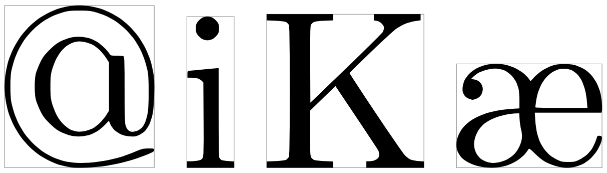
So why do these glyphs look somewhat different (smoother) from a normal outline Computer Modern? This is because the 600 may sound like a lot but actually isn’t. Distributed over an inch we have 600 pixels on 25.4 mm so roughly one pixel per 0.05 mm, which might be acceptable in a small print but not when scaled. Here is how an ‘m’ in 600 dpi pixels is coded (64 by 37 pixels):

```
0000001111110000000011111111000000000000000111111110000000000000
1111111111110000001111111111110000000000011111111111100000000000
1111111111110000111100000011111100000001111000000111111000000000
1111111111110001110000000001111110000011100000000011111100000000
0000111111110011100000000000111111000111000000000001111110000000
0000001111110111000000000000111111001110000000000001111110000000
0000001111111110000000000000111111011100000000000001111110000000
0000001111111100000000000000011111111000000000000000111111000000
0000001111111100000000000000011111111000000000000000111111000000
0000001111111000000000000000011111110000000000000000111111000000
0000001111111000000000000000011111110000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000001111110000000000000000011111100000000000000000111111000000
0000011111111000000000000000111111110000000000000001111111100000
1111111111111111110000011111111111111111110000011111111111111111
1111111111111111110000011111111111111111110000011111111111111111
1111111111111111110000011111111111111111110000011111111111111111
```

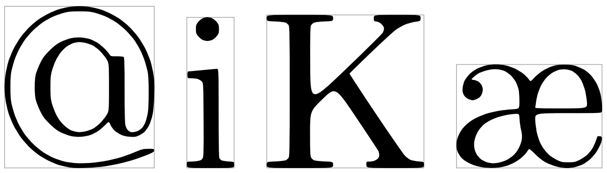
In the days of sending bitmaps to printers, possibly wrapped in a PostScript file, one could often recognize the characters from blobs like this. You might also remember some of that line printer art. At any rate, it shows that 600 dots per inch is much less than it sounds. A decent 2400 dpi (dots per inch) is more normal these days for printing on presses (with ink) but although there are 1200 dpi laser printers,

doi.org/10.47397/tb/45-1/tb139hagen-tracefonts

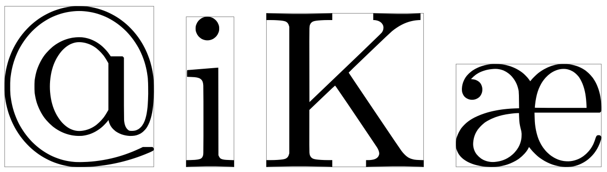
600 became the norm. After all, toner particles are not that small. For quite a while the authors used high speed OCE low temperature toner printer (first 508 dpi, later 600 dpi) and these were visually superior to most of what the competition had — but OCE never quite managed to do the same in color (only in lab testing, not reaching the market). Nowadays we use HP full width high speed inkjet printers that give a rather good quality full color experience at 600 dpi. This is what a scaled-up 2400 dpi bitmap look like:



This already looks more crisp although we can go back to the smoother variant by setting a higher threshold in potrace:



We can go higher. In the next rendering we use 7200 dpi bitmaps and now we see some details that didn't show up before. Keep in mind that subtle details might not be noticed in 10 point running text.



The top of the 'K' now has a little dent (likely not visible except with magnification). If you ever viewed a document with Latin Modern outlines you might recognize this. It is a side effect of outlines having that information while a bitmap needs to get the exact bits, which won't happen if such a dent stays beyond the pixel threshold. We now are ready for some real text. We define two font features to load bitmap and outlines, respectively:

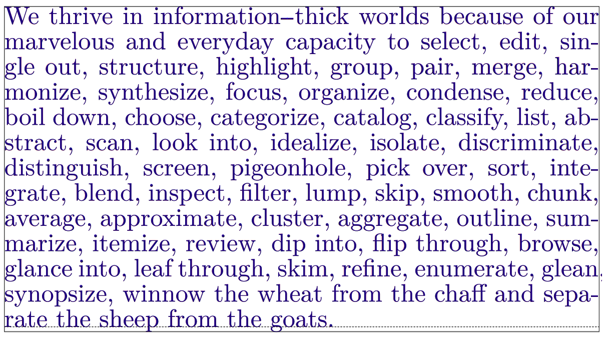
```
\definefontfeature[whateverpk][default]
 [reencode=ontarget-cmr.enc,bitmap=pk]
\definefontfeature[whateverpt][default]
 [reencode=ontarget-cmr.enc,bitmap=outline]
```

and some colors:

```
\definecolor[pkcolor][r=1,t=.5,a=1]
\definecolor[ptcolor][b=1,t=.5,a=1]
```

We use the following three font definitions in three overlaid examples (figure 1). The results are close enough to justify a closer look at the possibilities.

```
\definefont[PKdemoA]
 [file:lmroman10-regular.otf*default sa 1.2]
\definefont[PKdemoB]
 [file:ontarget-cmr10.tfm*whateverpk sa 1.2]
\definefont[PKdemoC]
 [file:ontarget-cmr10.tfm*whateverpt sa 1.2]
```



We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 1: Overlaid regular, bitmap and potraced text.

In figure 2 we see from top to bottom: Latin Modern OpenType outlines, a bitmap Computer Modern and a potraced Computer Modern. The fourth line has the bitmap and potraced overlaid. Figure 3 shows a small portion of the last one as seen on screen. Blown up, some drift is seen but so far we didn't find a way to get rid of it. Some is due to the way glyph streams get rendered and synchronized (after spacing, for instance).

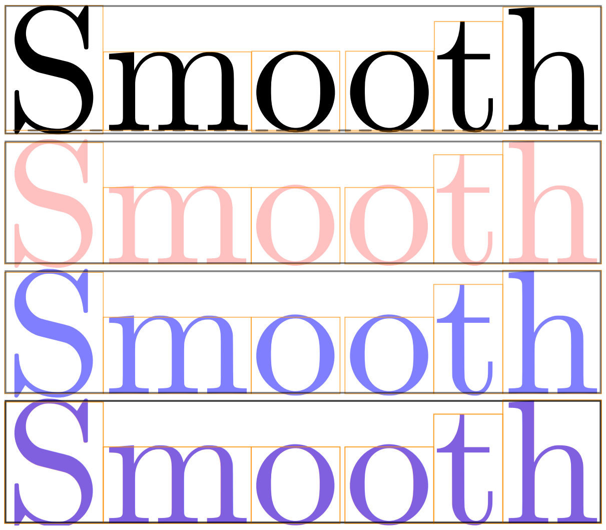


Figure 2: Bitmap and potraced compared; from top to bottom: OpenType, PK, potraced and overlaid.

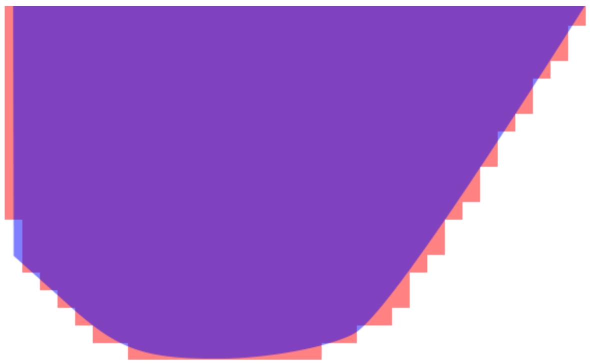


Figure 3: An enlarged clip of the overlay.

When not overlaid we get this for a normal Latin Modern Regular:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

And this for a Computer Modern Roman PK bitmap:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

The same Computer Modern Roman outlined by potrace gives this:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Because these are outlines we can scale them nicely with `\glyphscale 800` here:

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We can scale further, for instance with an extra `\glyphxscale 1200`. This can of course also be done with bitmaps but outlines are a safer bet.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

The conversion happens in the backend and can have some impact on the runtime but we cache the outlines, so a subsequent run is faster. Of course outlines are more efficient than bitmaps in terms of bytes and viewers also tend to render them better than bitmaps, which, especially at a lower resolution, can look pretty bad (in some viewers).

One might wonder if bitmaps are worse than outlines. When we use a reasonable resolution there is no need to generate more than one size, and in ConTEXt we assume this anyway because we scale all fonts to 10 big points and scale that shared instance on demand. The 600 dpi 'm' that we showed has 64 pixels in the horizontal direction. So, 75 of these (a line of text) need 4800 pixels, which is more than enough for a 4–8 display. Unfortunately viewers still render a bitmap font somewhat badly.

Let's dive a little into why bitmaps might render suboptimally in PDF viewers. Consider these three lines typeset in our three fonts:

```
\PKdemoA Smooth \vskip.1ex
\PKdemoB Smooth \vskip.1ex
\PKdemoC Smooth \vskip.1ex
```

These 'Smooth' lines (this time in 12pt) end up in the PDF file as follows :

```
BT
/F1 10 Tf
1.195514 0 0 1.195514 0 30.316793 Tm
 [<000100020003>-28<000300040005>] TJ
/F2 10 Tf
1.195514 0 0 1.195514 0 15.415656 Tm
 [<010203>-28<030405>] TJ
/F3 10 Tf
1.195514 0 0 1.195514 0 0.514763 Tm
 [<010203>-28<030405>] TJ
ET
```

This code first switches to font `/F1`, a wide outline font so we have four-byte indices (in angle brackets). Next we trigger `/F2`, the bitmap variant and finally the potraced `/F3`; these are both Type 3 fonts so they get two-byte indices. We don't scale except to the 10 big points design size. After such a switch comes lines of text and there we do scale, here by 1.195514 in both directions. We're slightly off 1.2 because the PDF font system (by tradition) is set up in PostScript (big) points so we need to scale up a little from 12pt to 12bp. Scaling an outline is translated (in the end) to some factor and the renderer can keep the device into account when it comes to rounding. With bitmaps it's different, because these are not mathematically-defined fonts, but some image that gets scaled. This can introduce the first inaccuracy. An inline bitmap in PDF is given between `ID` and `EI` operators, as demonstrated in the next two charproc entries for the Type 3 bitmap font (reformatted and abridged to save space):

```
21 0 obj
<< /Length 40708 >>
stream
556 0 55 -22 498 703 d1
q
442 0 0 725 55 -22 cm
BI
  /W 442 /H 725 /IM true /BPC 1 /D [1 0]
ID ...bytes...
EI
Q
endstream endobj
```

Notice the difference in the `/Length`:

```
22 0 obj
<< /Length 42784 >>
stream
833 0 33 0 809 440 d1
q
775 0 0 440 33 0 cm
BI
  /W 775 /H 440 /IM true /BPC 1 /D [1 0]
ID ...bytes...
EI
Q
endstream endobj
```

In contrast, a character in the potraced outline font looks like this:

```
31 0 obj
<< /Length 3678 >>
stream
556 0 55 -22 498 703 d1
q
1 0 0 1 55 -22 cm
172 723.96045697 m 144.84445441 720.6382363 ...
Q
endstream endobj
```

The length is much smaller and the outline shape is just a sequence of moveto (`m`), lineto (`l`) and curveto (`c`) operators mixed with numbers.

```
32 0 obj
<< /Length 4260 >>
stream
833 0 33 0 809 440 d1
q
1 0 0 1 33 0 cm
68.5 434.44174379 m 32.2 431.50947593 1.9375 ...
Q
endstream endobj
```

Experiments demonstrated that it's better to use rounded widths because otherwise (at least in SumatraPDF) we get some accumulated drift. The bitmap variants have a transform matrix like this:

```
442 0 0 725 55 -22 cm
775 0 0 440 33   0 cm
```

and the potraced outlines have:

```
1 0 0 1 55 -22 cm
1 0 0 1 33   0 cm
```

This means that a bitmap again gets scaled, luckily by an integer, but still there is some inaccuracy. In the end, we get the bits put on screen and especially at small scales we end up with artifacts in positioning and scaling. Where the font renderer is optimized for (indeed) rendering fonts, the bitmap renderer isn't. In figure 4 we see bitmaps being rendered bolder when they become smaller, because in the end, even at high resolutions, we're not talking pixels but bits (that can occupy multiple pixels). Outline fonts talk pixels, bitmap fonts speak in bits.

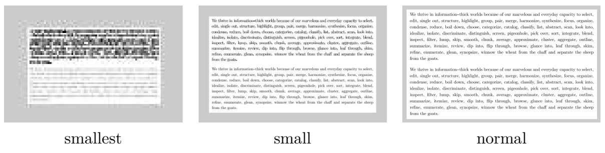


Figure 4: Three zoom levels compared.

We haven't yet discussed how we got the bitmaps that we used. You might have noticed in the examples that there are some differences with the outline when it comes to dimensions. This is partly due to the fact that where Latin Modern is an OpenType font with no limits to dimensions, Computer Modern has to accommodate the limited number of heights, depths and widths that the TFM format permits. Think of arbitrary values of height compared to categories of height.

When you generate a bitmap you rely on scripts that do the work and these work together with so-called printer modes as defined in the METAFONT file `modes.mf` (`https://ctan.org/pkg/modes`). These modes are for printers which means that there can be compensation going on: rounding up or down of points exceeding bounding box edges, the size of printer pixels (toner, ink) and accuracy of positioning them, etc. In our case, when we went for 8000 dpi we ended up with a device that did more compensation than needed. Better is to investigate time in figuring out how to control the machinery to cook up (maybe) 7200 dpi bitmaps because down the inclusion route there is some division by 72. If we decide to play a bit more, we might as well first figure out how to control the bitmap generation and see if we can come up with this 7200 resolution. Not only does it divide nicely by 72 (for display) but also by 600 (for the average printer). To what extent that matters is to be seen.

A bitmap font (instance) is generated by METAFONT and driven by a (printer) mode defined in `modes.mf`. We added this one:

```
mode_def potrace =
    mode_param (pixels_per_inch, 2 * 3600) ;
    mode_param (blacker, 0) ;
    mode_param (fillin, 0) ;
    mode_param (o_correction, 1) ;
    mode_common_setup_ ;
enddef;
```

The `2 * 3600` is a trick to get around METAFONT maxing out at 4096 but internally being capable to deal with larger numbers. Of course we forgot to run `fmtutil-sys --byfmt mf` which is needed to get these modes in the format file, but eventually we managed to generate the 7200 dpi PK file for `cmr10`. Generating is easiest done with pdfTEX with `\pdfmapfile {}`, which wipes the mapping to a Type 1 file.

We mentioned using MetaPost in our first attempts to get an idea how well potrace can vectorize the bitmaps. Here is how that is done:

```
\startluacode
  local f = fonts.handlers.tfm.readers.loadpk
              ("cmr10.pk")
  if f then
    local g = f.glyphs[string.byte("R")]
    if g then
      local b
        = fonts.handlers.tfm.readers.showpk(g)
      potrace.setbitmap("demo",b)
    end
  end
\stopluacode

\startMPpage[offset=1ts]
    draw lmt_potraced [
        stringname = name,
        value      = "1",
    ];
\stopMPpage
```

In figure 5 we demonstrate three such renderings. Watch how the number of points increases as the shape gets better. In figure 6 we show the potraced variant alongside the OpenType outline. Left is the default potrace output, next comes the regular OpenType (here CFF) outline, and at the right we see two potraced results, both with `optimize = true` passed; the first has the default tolerance of 0.2 and

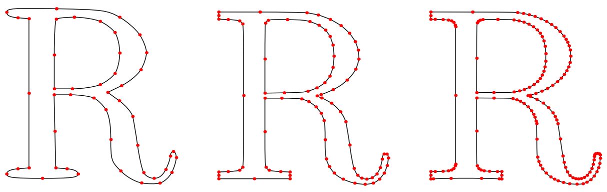


Figure 5: Using MetaPost for analysis.

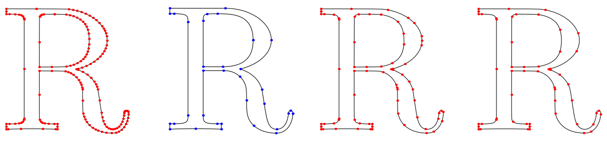


Figure 6: Comparing potraced bitmaps with Type 1.

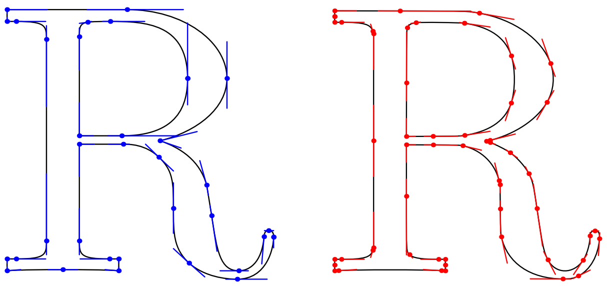


Figure 7: Comparing control points.

the second uses 0.5 and thereby has less points. For sure the middle one has less points which is nice, but the potrace ones are not that excessive so we can live with it. We've run into cases (especially math) where the regular outlines are also not perfect. Most will go unnoticed anyway given the small size at which glyphs are normally rendered. When you look closely at the rightmost output you'll notice that the bend in the stems makes for more points which is an indication that the METAFONT output might actually be more subtle. In figure 7 we also show the controls and you see subtle differences in the angles there. Also note that when there are no lines that indicates that there is likely a lineto instead of a curveto.

If you like the look of these shapes, take a look at Volume E of Don Knuth's "Computers & Typesetting" series. An incredible amount of work went into the details of the fonts and you'll run into brackets, beaks, crisps, notches, slabs, juts, dishes and more elements and parameters that are used. For instance the serifs as seen in the 'R' are actually made programmatically (so that they can be discarded in the sans shapes). If you check out the proof sheet of the 'R' you will only see the basic points that describe the character, not the points we see above, after conversion to (any kind of) outline.

So now that we can turn a PK into a proper outline we're done, right? Well, not entirely. Because we have relatively simple shapes (moveto, lineto and curveto) we can directly go to CFF and avoid the MetaPost to Type 3 conversions. Because we already can load (and adapt) CFF outlines it is not that complicated to do the reverse. The backend already can include them so we can also borrow code there.

When going from potrace output to CFF, we need a high resolution, so we started out again with 7200 dpi. Although in the end we were quite satisfied with the results, we tested with 20000 dpi and thought it looks even better than the Latin Modern successor (although one can argue about it). Some first experiments showed that it was doable but it actually took a whole day to (test and) decide how to get better results. For instance, MetaPost uses floats while a renderer uses integers, so we run into rounding issues if we delegate that (although we can include floats in CFF, it is not a success). When overlaying the MetaPost output and the CFF shapes the latter was quite disappointing: weird protrusions and bubbles. Of course one may doubt the implementation, but double and triple checking showed that we use the same numbers.

The results improved a lot when the results from potrace were multiplied by 10 (or 20) effectively giving very huge glyphs (on a 10000 unit canvas) and in the page stream scaling down by that factor. By then using rounded values we got enough precision to get the CFF results close to the (always) high quality MetaPost rendering. In figure 8 we can see how close they became.

The next question is, how do we control this: PK, MetaPost Type 3 or native CFF? Even more challenging is that we had wanted to use different resolutions, and mix these three methods, if only because we want to be able to experiment and document the mix. That means that it has to be available not only in the font feature mechanism, which is rather trivial, but also in the backend, which then involves a font with the same name (say CMR10) to be rendered differently, so we need different handlers, distinctive caching of streams, etc. In the end we got there. It is even possible to mix variants with different potrace parameters in one document.

We start with CFF definitions. In figure 9 we show three resolutions overlaid, with the 7200 dpi variant below. In figure 10 we show the default and optimized (fewer points) traces.

```
\definefontfeature[CMRCFF]
  [reencode=ontarget-cmr.enc,bitmap=cff]
```

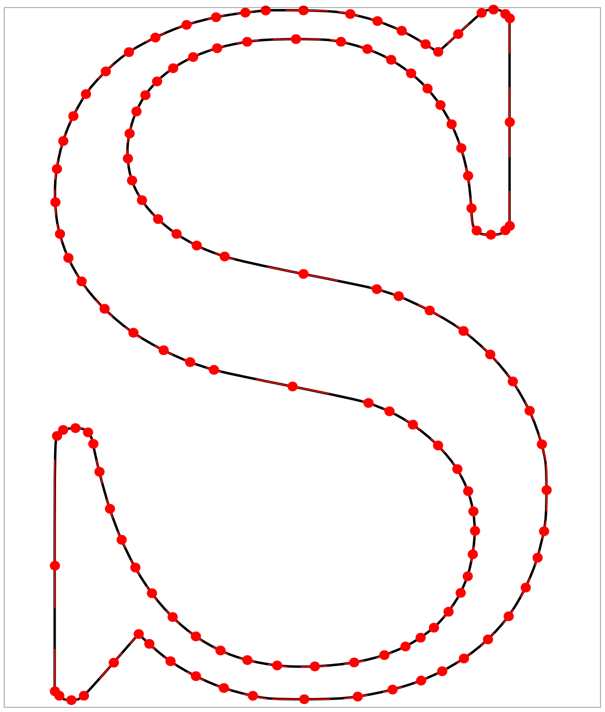


Figure 8: A MetaPost rendering on top of its CFF counterpart.

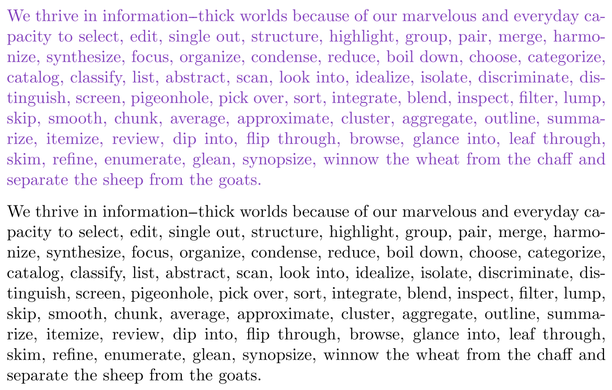
We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 9: Above: 7200, 2400, 600 dpi CFF variants overlaid; bottom: cf. 7200 dpi CFF variant alone.

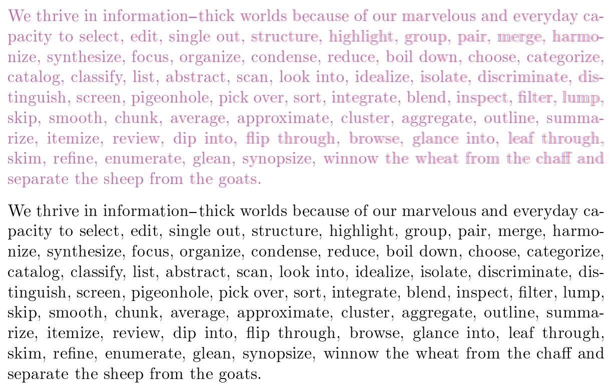
```
\definefontfeature[MyFontCffA]
  [default,CMRCFF][resolution=7200]
\definefontfeature[MyFontCffB]
  [default,CMRCFF][resolution=2400]
\definefontfeature[MyFontCffC]
  [default,CMRCFF][resolution=600]
\definefontfeature[MyFontCffD]
  [default,CMRCFF][resolution=7200,
                   potrace={optimize=true}]
```

The overlays look fuzzy, demonstrating the need for high resolutions. Font definitions are done as follows; we only show one definition:


We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 10: Above: Normal and optimized 7200 dpi variants overlaid; below: cf. 7200 dpi variant alone.


We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

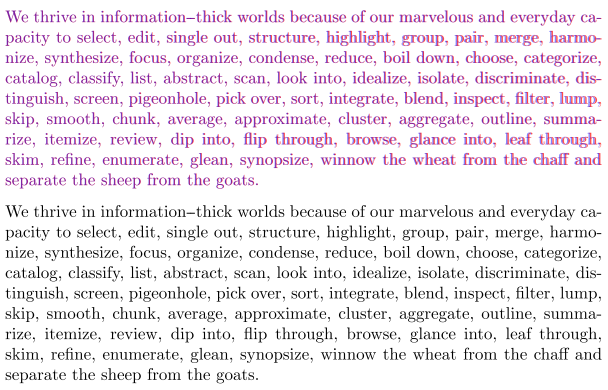
Figure 11: Above: 7200, 2400, 600 dpi PK variants overlaid; below: cf. 7200 dpi PK variant alone.

```
\definefont[MyFont]
  [file:ontarget-cmr10.tfm*MyFontCffA]
```

These definitions are used in figure 11. The differences aren't immediately apparent in small print but zoom in (if you're online) and you'll understand why we need more than 600 dpi to feel comfortable.

Finally we show the MetaPost-generated threesome in figure 12 and these look quite reasonable. One thing to keep in mind when wrapping shapes into a Type 3 font is that one has to make sure that color keeps working.

So, how useful is all this? Maybe it's time for a revival of METAFONT. Or maybe we can get some old designs out of the archives where they got tagged obsolete and use them again. Or maybe it's just for the fun of it. We started out with PK bitmaps that we need to support anyway. Next we were curious how well a traced outline would look, and for that using a MetaPost Type 3 font makes sense. Then we took the challenge to turn potrace output into a CFF OpenType font. We could now make a whole tool chain but it makes little sense: we can do it in ConTEXt, so we're fine, and we don't expect widespread usage


We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 12: Above: 7200, 2400, 600 dpi MetaPost variants overlaid; below: cf. 7200 dpi MP variant alone.

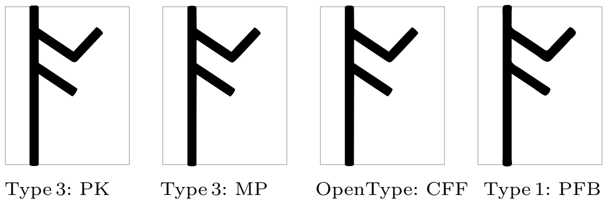


Figure 13: A test with one of the allrunes fonts. Left is using the PK at a 7200 dpi resolution. Next is the potraced 7200 PK original outline. Third from left is the generated CFF. Right is the shipped PFB.

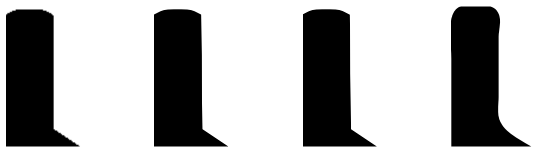


Figure 14: Enlarged clips of the variants in figure 13.

outside the TEX ecosystem. Next on the agenda is to locate some interesting METAFONTs and see if they can be put to use. In case you wonder how these eight bit fonts fit into the Unicode ecosystem, don't worry, we can use the virtual font mechanism to hook shapes into existing fonts (which is what we do anyway) or create combined fonts. We can even make variable fonts using METAFONT, but for that we need to become experienced designers first. As a teaser we show a character from the runes font by Carl-Gustav Werner in figures 13 and 14. Mikael, knowing the author, promised to come up with a proper encoding for that one, so stay tuned.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University
`mickep (at) gmail dot com`

Basic Latin brevigraphs listed in *Polonia Typographica Saeculi Sedecimi* — progress report

Janusz S. Bień

1 Introduction

The fonts of several 16th century printers active in Poland, namely Aleksander Augezdecki, Jan Haller, Kasper Hochfeder, Florian Ungler (the first and second printing house) and Maciej Wirzbięta, have been described in the series of 12 fascicules entitled *Polonia Typographica Saeculi Sedecimi* published in years 1959–1981.[1] Almost all of them are digitized but available only for "digital lending" in ACADEMICA[2] because, I surmise, it's not clear who owns the copyright as this was a collaborative effort of several persons (only one contributor is still alive) and institutions.

In the fascicules every font is illustrated by an excerpt of a text and sometimes additionally by a table of the character set; an example of such a table is presented in Fig. 1. Most of the tables have been prepared by Maria Błońska with some help from Anna Wolińska and Henryk Bułhak (the editor of several fascicules); some tables were prepared by Anna Śliwa, Alodia Kawecka Gryczowa (also the editor of several fascicules) and Paulina Buchwald-Pelcowa (the editor of the whole series). The number of font tables is over seventy and the total number of glyphs in the tables is over six thousand.

Unfortunately I missed the opportunity to get first-hand information on how the tables were prepared when talking by phone with Paulina Buchwald-Pelcowa in 2022 (she died two years later). I got some rudimentary information from Henryk Bułhak, also in phone calls, but he was able to provide me only with rather general information: the glyphs were cut out with a razor blade from photocopies and pasted together. This information seems relevant because it shows what kind of mistakes can be expected in the tables: if a character occurs in a table then it can be displaced or misassigned (see sec. 7), but definitely exists in a text; on the other hand, some omissions are possible. For example, the glyph in Fig. 2 is not listed in the table in Fig. 1, but can be found in the texts printed reportedly with this font; according to Peter Baker, the meaning is *cis*.[3]

[1] The first two fascicules were published in 1936 and 1937, but we are interested in their second revised editions because these were supplemented by the character set tables.

[2] *Interlibrary loan system of books and scientific publications*: `https://academica.edu.pl/`

[3] `github.com/psb1558/Junicode-font/discussions/255`

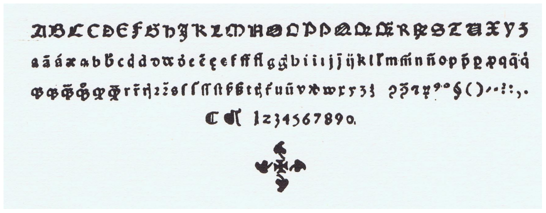


Figure 1: Ungler's second printing house font no 16 (a fragment of Plate 359)

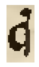


Figure 2: The letter or the ligature *cis*? Cf. Fig. 40.

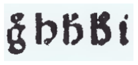


Figure 3: A fragment of Plate 168. The fourth glyph is interpreted as *h*, not a *b*, because of its position in the font table. See sec. 8

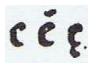


Figure 4: A fragment of Plate 357. The last glyph is interpreted as *e* with ogonek because of its position in the font table.

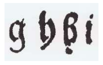


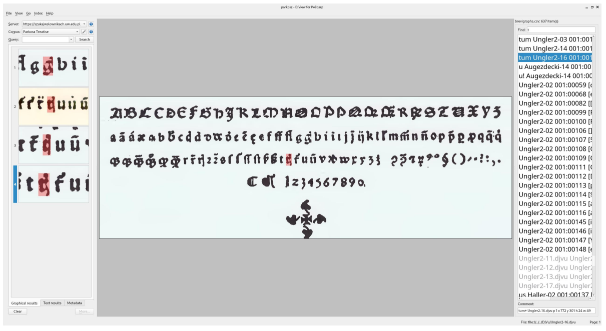
Figure 5: A fragment of Plate 411. The third glyph is interpreted as *h* because of its position in the font table.

No comments to the tables are provided, but the order of glyphs is sometimes relevant for their interpretation (see Figures 3, 4, 5).

The quality of the glyph images is sometimes quite low; I understand no better samples were available.

Early prints used many abbreviations which were the descendants of the abbreviations used in manuscripts; we discuss here a subset of them, called brevigraphs. Quite often they consisted of a regular non-modified letter supplemented by a diacritical mark, usually a macron or a similar glyph, placed above. It is natural to call them composed brevigraphs. On the other hand there are abbreviations in a shape of a modified letter or a letter-like symbol; we call them basic brevigraphs even if they are accompanied by a diacritical mark.

The work described here consisted primarily of creating computer indexes to allow comparison of similar characters from the same or different fonts



```
brevigraphs.csv: 637 item(s)
Find: t
tum Ungler2-03 001:001
tum Ungler2-14 001:001
tum Ungler2-16 001:001
u Augezdecki-14 001:00
u! Augezdecki-14 001:00
Ungler2-02 001:00059 [
Ungler2-02 001:00068 [
Ungler2-02 001:00082 [
Ungler2-02 001:00099 [
Ungler2-02 001:00100 [
Ungler2-02 001:00106 [
Ungler2-02 001:00107 [
Ungler2-02 001:00108 [
Ungler2-02 001:00109 [
Ungler2-02 001:00111 [
Ungler2-02 001:00112 [
Ungler2-02 001:00113 [
Ungler2-02 001:00114 [
Ungler2-02 001:00115 [
Ungler2-02 001:00116 [
Ungler2-02 001:00145 [
Ungler2-02 001:00146 [
Ungler2-02 001:00147 [
Ungler2-02 001:00148 [
Ungler2-11.djvu Ungler
Ungler2-12.djvu Ungler
Ungler2-13.djvu Ungler
Ungler2-17.djvu Ungler
us Haller-02 001:00137 [
Comment:
tum+ Ungler2-16.djvu p 1 x 772 y 301 h 24 w 49
File: file://../../DjVu/Ungler2-16.djvu    Page: 1
```

Figure 6: Comparing characters with djview4poliqarp

(Fig. 6). The indexes and other resources are available in a public repository;[4] the repository site provides *Issues* and *Discussions* tabs for reporting mistakes and giving comments.

The paper is considered a progress report for two reasons. First, the character indexes should be supplemented by indexes showing their meaning and their use in texts, in a similar way as described in [5]. Secondly, I omitted some interesting glyphs because I was not sure how to interpret them. An example is presented in Fig. 7: is the last but one glyph a modification of the letter *h*, or is this just the letter *h* with a diacritic mark which happens to touch the letter?

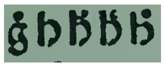
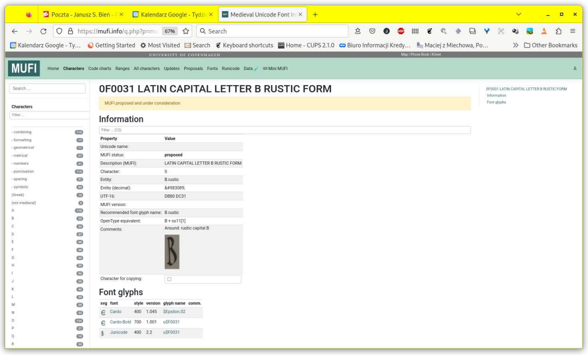


Figure 7: A fragment of Plate 21. Is the last but one glyph a basic brevigraph?

An important question for every basic brevigraph is whether it has been assigned a codepoint in the Unicode standard.[5] Checking the character charts is unfortunately not sufficient for two reasons. First, the character name is not intended to provide the full information about the character, it should be treated as a more or less arbitrary label. Secondly, in principle (the practise is sometimes questionable) the standard defines characters, not glyphs, and the glyph in a chart is only one of the possible representations of the character (an example is given in sec. 9). In consequence it is useful to also look up the character proposals and related documents in the Unicode Technical Committee Document Registry[6] (a similar resource is the ISO/IEC JTC1/SC2/WG2 register[7]). It is also useful to look for alternative glyphs in specialized fonts (see sec. 12).

[4] `github.com/jsbien/early_fonts_inventory`
[5] `home.unicode.org/`
[6] `unicode.org/L2/`
[7] `unicode.org/wg2/`



0F0031 LATIN CAPITAL LETTER B RUSTIC FORM

MUFI proposed and under consideration

Information

Property	Value
Unicode name:	
MUFI status:	proposed
Description (MUFI):	LATIN CAPITAL LETTER B RUSTIC FORM
Character:	B
Entity:	B.rustic
Entity (decimal):	&#983089;
UTF-16:	DB80 DC31
MUFI version:	
Recommended font glyph name:	B.rustic
OpenType equivalent:	B + ss11[1]
Comments:	Ansund: rustic capital B
Character for copying:	

Font glyphs

svg	font	style	version	glyph name	comm.
	Cardo	400	1.045	$Epsilon.02	
	Cardo-Bold	700	1.001	u$F0031	
	Junicode	400	2.2	u$F0031	

Figure 8: MUFI LATIN CAPITAL LETTER B RUSTIC FORM assignment proposal as of 2024-01-01

Another interesting question is whether the brevigraph has been assigned a codepoint in the Unicode Private Use Area by the Medieval Unicode Font Initiative.[8] The MUFI assignments at first, up to version 4.0, were published as recommendations in the form of PDF documents [11]. They list over 1500 pure Unicode and Private Use Area characters in the Latin alphabet of potential use for the encoding of old text sources.

Nowadays, the recommendations have the form of a database which can be browsed online (Fig. 8). For some time a subset of the data content is also available for download, under the Creative Commons Attribution-ShareAlike 4.0 license, in the form of a CSV or JSON file.

We will use here also the resources of *Projet d'Inventaire des Caractères Anciens*[9] created and maintained by Jacques André.

Last but not least, it is important whether a brevigraph can be rendered adequately by a font. Our primary focus is on Peter Baker's Junicode Two font, as it is available under a free license[10] and contains some characters not available elsewhere [7]. We also use George Douros' Symbola font[11] for some characters not available in Junicode.

It is worth mentioning that some of the brevigraphs discussed here were used in Gutenberg's bibles. The character set of these books has been the subject of several publications; they are referenced in [2] and [6]. I also 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.)

[8] `mufi.info/`
[9] `jacques-andre.fr/PICA/`
[10] `github.com/psb1558/Junicode-font`
[11] `dn-works.com/ufas/`

2 The workflow

It is an old idea of mine to use the fact that for compression purposes the identical or similar shapes are collected into shape dictionaries. It seems that this approach, named mixed raster content, is used now in JPEG2000, but the first format to use it successfully was DjVu.

I designed two tools which are based on this approach. The first one was a quick and dirty modification of a standard DjVu viewer (Fig. 9). It was originally implemented by Michał Rudolf twelve years ago, with important contributions made later by Alexander Trufanov.[12] It is quite good for getting a quick overview of the shapes in a document, but it is not convenient enough for analysing them in detail.[13]

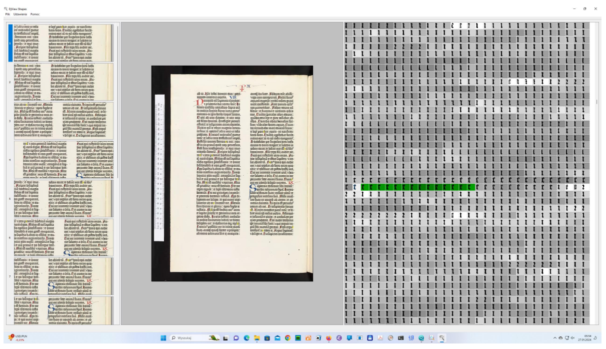


Figure 9: djview-shapes and Gutenberg's bible

The second tool was a sophisticated client-server system. The idea was that a database will store shapes from different documents provided by different persons or institutions and accessed remotely by interested users. The shapes were exported to a MySQL database. Unfortunately the client[14] was a complete failure, since due to some wrong coding decisions it was prohibitively slow. There was neither opportunity nor sufficient motivation to reimplement it in a better way.

So when working on the present paper my main tool was Alexander Trufanov's `djvudict` program[15] which dumps a DjVu shape dictionary in an almost human-readable form, despite the fact that the program does not seem to be fully reliable (e.g., for some not yet known reasons some shapes are skipped).

The first step was accessing the scans of *Polonia Typographica* and preparing (with Gimp) the images of the relevant tables. Then the images were converted to DjVu (with Friedric Foebel's Python 3 fork of didjvu[16]) and supplemented by appropriate metadata. Next they were processed by djvudict. The primary DjVu file names have a form like `Augezdecki-01a_PT08_403.djvu`, where `Augezdecki` is the name of the printer, `01a` is the font number sometimes supplemented by a letter, `PT08` is the identification of the *Polonia Typographica* fascicule and `403` is the plate number (they are numbered continuously in the whole series; many contain woodcuts of no interest to us). The djvudict output is placed in the directories with shorter names, as in `Augezdecki-01a`.

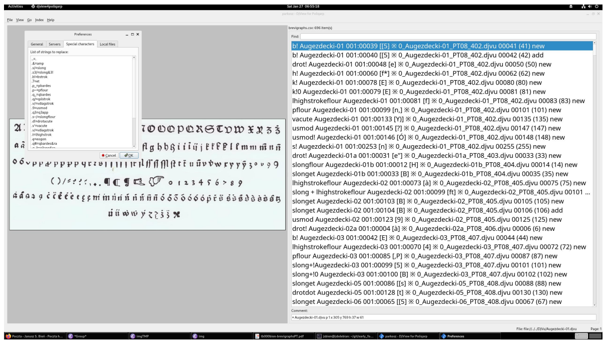
A quick and dirty Python program (written, or rather put together from pieces of code found on the Internet, by myself) converts the djvudict output to an index for the djview4poliqarp program (described already in [5] and [8]); the shape identifiers are preserved. The index contains also the results of OCR processing done with *Tesseract*, but at present, due to the lack of appropriate training, they are of essentially no use. The file names are in the form `Augezdecki-01a.csv`.

The indexes unfortunately require some hand editing with djview4poliqarp. The first stage is to create an index named like `Augezdecki-01a_tmp.csv`, where the interesting elements are marked with '`+`' in the so-called comment field. Entries are marked with '`#`' when there is a need to adjust the bounding box; it is not yet clear why this is sometimes needed. Entries marked with '`^`' also require adjusting the bounding box, but for a different reason: the shapes recognized by the DjVu compression algorithm are just connected components, so diacritics are usually separate objects.

The files `*_tmp.csv` are processed with grep to put the marked entries into the indexes named `*_work.csv`, where the bounding boxes are adjusted if needed. The files form the basis for the intermediate brevigraph indexes named `*_workbr.csv` where the entries are supplemented by the brevigraph names.

The brevigraph names serve a purely technical goal: they allow grouping similar brevigraphs together in the djview4poliqarp program. However, the choice of the names is not obvious. The official Unicode names and the Unicode-like MUFI names are cumbersome because of their length, e.g., LATIN CAPITAL LETTER V WITH DIAGONAL STROKE. As an alternative, I was considering using names derived from the XML entity names provided by MUFI, also for selected pure Unicode characters. Some are

[12] `github.com/jsbien/djview4shapes`
[13] When the present paper was almost finished, some changes were made to the program which make it much more useful.
[14] `github.com/jsbien/ndt/wiki/z_shapes`
[15] `github.com/trufanov-nok/djvudict`
[16] `github.com/FriedrichFroebel/didjvu`



```
b! Augezdecki-01 001:00039 [[5] ※ 0_Augezdecki-01_PT08_402.djvu 00041 (41) new
b! Augezdecki-01 001:00040 [[5] ※ 0_Augezdecki-01_PT08_402.djvu 00042 (42) add
drot! Augezdecki-01 001:00048 [e] ※ 0_Augezdecki-01_PT08_402.djvu 00050 (50) new
h! Augezdecki-01 001:00060 [f*] ※ 0_Augezdecki-01_PT08_402.djvu 00062 (62) new
k! Augezdecki-01 001:00078 [E] ※ 0_Augezdecki-01_PT08_402.djvu 00080 (80) new
k!0 Augezdecki-01 001:00079 [E] ※ 0_Augezdecki-01_PT08_402.djvu 00081 (81) new
lhighstrokeflour Augezdecki-01 001:00081 [f] ※ 0_Augezdecki-01_PT08_402.djvu 00083 (83) new
pflour Augezdecki-01 001:00099 [n,] ※ 0_Augezdecki-01_PT08_402.djvu 00101 (101) new
vacute Augezdecki-01 001:00133 [Y)] ※ 0_Augezdecki-01_PT08_402.djvu 00135 (135) new
usmod Augezdecki-01 001:00145 [?] ※ 0_Augezdecki-01_PT08_402.djvu 00147 (147) new
usmod! Augezdecki-01 001:00146 [Ó] ※ 0_Augezdecki-01_PT08_402.djvu 00148 (148) new
s! Augezdecki-01 001:00253 [n] ※ 0_Augezdecki-01_PT08_402.djvu 00255 (255) new
drot! Augezdecki-01a 001:00031 [e"] ※ 0_Augezdecki-01a_PT08_403.djvu 00033 (33) new
slongflour Augezdecki-01b 001:00012 [H] ※ 0_Augezdecki-01b_PT08_404.djvu 00014 (14) new
slonget Augezdecki-01b 001:00033 [B] ※ 0_Augezdecki-01b_PT08_404.djvu 00035 (35) new
lhighstrokeflour Augezdecki-02 001:00073 [à] ※ 0_Augezdecki-02_PT08_405.djvu 00075 (75) new
slong + lhighstrokeflour Augezdecki-02 001:00099 [ft] ※ 0_Augezdecki-02_PT08_405.djvu 00101 ...
slonget Augezdecki-02 001:00103 [B] ※ 0_Augezdecki-02_PT08_405.djvu 00105 (105) new
slonget Augezdecki-02 001:00104 [B] ※ 0_Augezdecki-02_PT08_405.djvu 00106 (106) add
usmod Augezdecki-02 001:00123 [9] ※ 0_Augezdecki-02_PT08_405.djvu 00125 (125) new
drot! Augezdecki-02a 001:00004 [à] ※ 0_Augezdecki-02a_PT08_406.djvu 00006 (6) new
b! Augezdecki-03 001:00042 [E] ※ 0_Augezdecki-03_PT08_407.djvu 00044 (44) new
lhighstrokeflour Augezdecki-03 001:00070 [4] ※ 0_Augezdecki-03_PT08_407.djvu 00072 (72) new
pflour Augezdecki-03 001:00085 [,P] ※ 0_Augezdecki-03_PT08_407.djvu 00087 (87) new
slong+!Augezdecki-03 001:00099 [5] ※ 0_Augezdecki-03_PT08_407.djvu 00101 (101) new
slong+!0 Augezdecki-03 001:00100 [B] ※ 0_Augezdecki-03_PT08_407.djvu 00102 (102) new
slonget Augezdecki-05 001:00086 [[s] ※ 0_Augezdecki-05_PT08_408.djvu 00088 (88) new
drotdot Augezdecki-05 001:00128 [t] ※ 0_Augezdecki-05_PT08_408.djvu 00130 (130) new
slonget Augezdecki-06 001:00065 [[5] ※ 0_Augezdecki-06_PT08_408.djvu 00067 (67) new
```

```
+ Augezdecki-01.djvu p 1 x 305 y 769 h 37 w 61
File: file://../../DjVu/Augezdecki-01.djvu    Page: 1
```

Figure 10: Keyboard shortcuts in djview4poliqarp

short and mnemotechnical, e.g. `&pbardes;` (U+A751 latin small letter P with stroke, here BAR, through DEScender). Some are short and not mnemotechnical, e.g., `&q3app;` (U+E8B4 LATIN SMALL LETTER Q LIGATED WITH FINAL ET; 3 may suggest the shape of the final *et*, but I have no association to suggest for `app`), some are mnemotechnical but rather long, e.g. `&lhighstrok;` (U+A749 latin small letter L with HIGH STROKE).

Nevertheless, after some hesitation, I decided to use those names (with '`&`' and '`;`' stripped) for my purposes. The crucial factor in making this decision was the fact that djview4poliqarp has a kind of macro facility (Fig. 10). The configuration file[17] has an `[edit]` section which can contain appropriate settings.

For characters which are present neither in Unicode nor in MUFI I use *ad hoc* names. Characters which are difficult to identify I handle in an analogical way. Such names often don't identify the glyphs uniquely, but merely point to similar glyphs.

For technical reasons the names are placed first in the comment field and then moved to the entry field with a program. The final indexes have the names in the form of `*_br.csv`. An aggregated index is also created which is named simply `brevigraphs.csv`.

The figures in the present paper were prepared in a way similar to that used for [8]: a program converts the index of the selected glyphs into the expex[18] code and creates a set of graphic snippets from the DjVu documents.

The glyphs in the figures are numbered for reference purposes and accompanied by the self-explanatory abbreviations of printing house names and font numbers.

[17] `~/.config/djview-poliqarp/djview-poliqarp.conf`
[18] `ctan.org/pkg/expex`

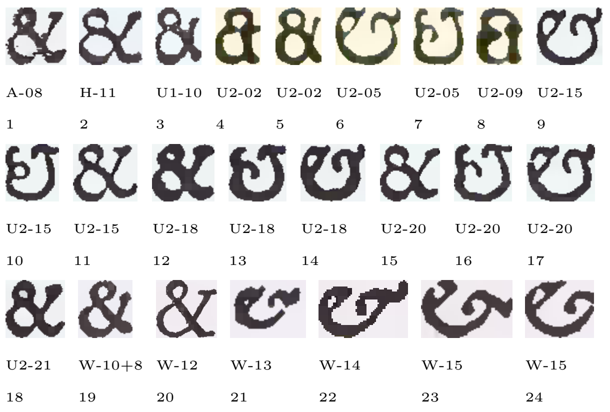

A-08 H-11 U1-10 U2-02 U2-02 U2-05 U2-05 U2-09 U2-15
1 2 3 4 5 6 7 8 9
U2-15 U2-15 U2-18 U2-18 U2-18 U2-20 U2-20 U2-20
10 11 12 13 14 15 16 17
U2-21 W-10+8 W-12 W-13 W-14 W-15 W-15
18 19 20 21 22 23 24

Figure 11: Ampersand

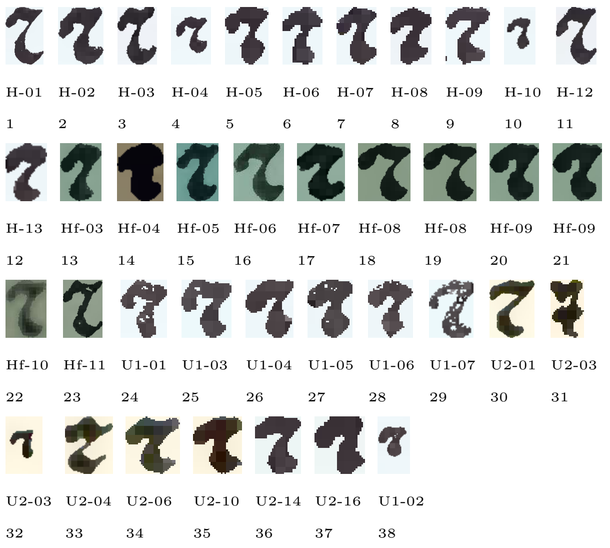

H-01 H-02 H-03 H-04 H-05 H-06 H-07 H-08 H-09 H-10 H-12
1 2 3 4 5 6 7 8 9 10 11
H-13 Hf-03 Hf-04 Hf-05 Hf-06 Hf-07 Hf-08 Hf-08 Hf-09 Hf-09
12 13 14 15 16 17 18 19 20 21
Hf-10 Hf-11 U1-01 U1-03 U1-04 U1-05 U1-06 U1-07 U2-01 U2-03
22 23 24 25 26 27 28 29 30 31
U2-03 U2-04 U2-06 U2-10 U2-14 U2-16 U1-02
32 33 34 35 36 37 38

Figure 12: Tironian note *et*

3 Non-alphabetic brevigraphs

Figure 11 shows ampersand, the brevigraph which in one of its forms has survived to the present time; it is a very old abbreviation of the word *et*. It has two forms, both of them are available in the Junicode family of fonts: '&' (Junicode-Regular) and '*&*' (Junicode-Italic). In computer code the first form has been available at least since ASCII (*American Standard Code for Information Interchange*), which was created in 1963. The Unicode charts also show only the first form.

The brevigraph presented in Fig. 12 is without any doubt the Tironian note *et* (Tironian notes are named after Tiro, the secretary of Cicero, who is credited with inventing them), used always as a separate word. The brevigraph is present in Unicode since version 3.0 (published in 1999) as U+204A `TIRONIAN SIGN ET` with the canonical glyph '⁊'. The Junicode

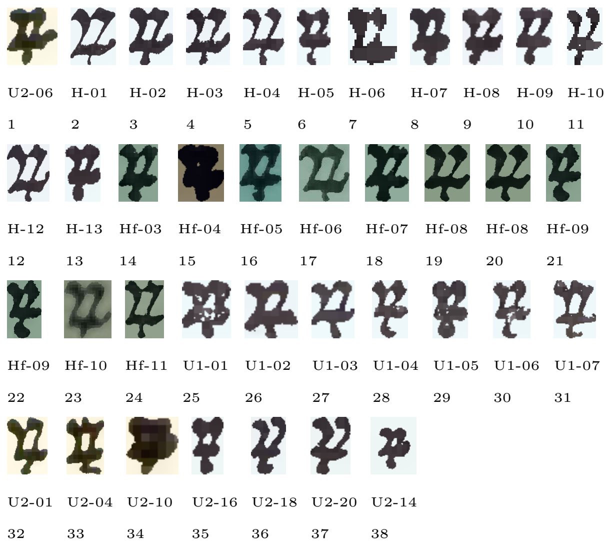


Figure 13: The letter *rum* rotunda

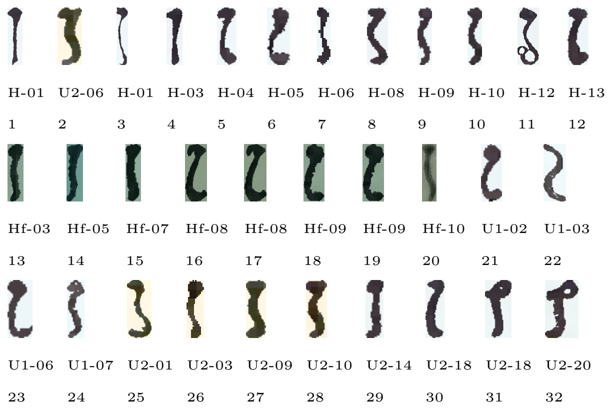


Figure 14: The letter *is*

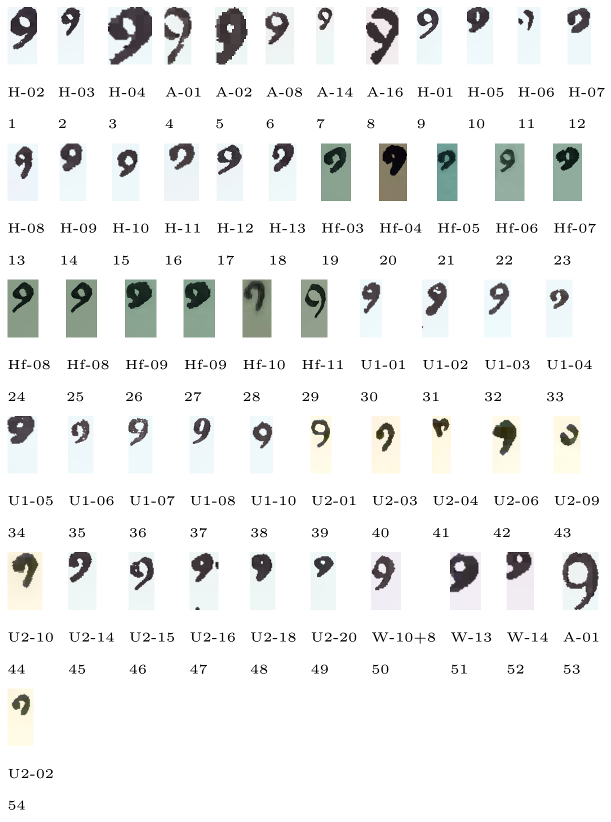


Figure 15: The letter *us*

font has also another variant of the glyph, namely ‘⁊’ (accessed with OpenType character variant feature or just with the code U+F001D), which is quite close to the shape of most glyphs in Fig. 12. Item 31 in the figure is yet another variant, called in the font manual *Tironian et sign later form with bar*; it is available in Junicode with OpenType feature *ss10* and the tags ‘a’‘1’: ‘ƻ’.

The brevigraph presented in Fig. 13 is present in Unicode since version 5.1.0 (published in 2000) as U+A75D LATIN SMALL LETTER RUM ROTUNDA with the canonical glyph ‘ꝝ’, which is quite close to the glyphs in the figure. It can mean *-rum* or *-rom*. Although the name may suggest that this is a variant of the letter *rum*, their shapes have little in common (see sec. 12).

As noted in [1, p. 130], Unicode has additionally two similar symbols: U+0264 JUPITER, U+1F729 ALCHEMICAL SYMBOL FOR TIN ORE. In the Symbola font these three characters look like, respectively: ꝝ, ♃ and 🜩. They look different, but this is the decision of the contemporary font designer. The glyphs listed in Fig. 13 could probably represent any of those three characters; this has to be checked in the original texts.

All the glyphs in Fig. 14 are in my opinion variants of the character U+A76D LATIN SMALL LETTER IS, present in Unicode since version 5.1.0 (published in 2000) with the canonical glyph ‘ꝭ’. Of course their usage should be verified in the original texts.

The glyphs in Fig. 15 looking like the digit *9* are instances of the well-known brevigraph present in Unicode as U+A770 MODIFIER LETTER US with representative glyph ‘ꝰ’; it was introduced in version 5.1.0 (published in 2008), and *modifier* means it is not on the baseline. It is used always at the end of words. The meaning of the glyphs similar to a circle, like item 43, is to be checked in the texts, as it can be just a raised small letter *o* (in Unicode, U+1D52 MODIFIER LETTER SMALL O).

The base glyphs in Fig. 16, despite a slightly different shape, can be identified with the character called by MUFI LATIN ABBREVIATION SIGN SMALL CON [11, s. 29] and in Unicode unified with U+2184

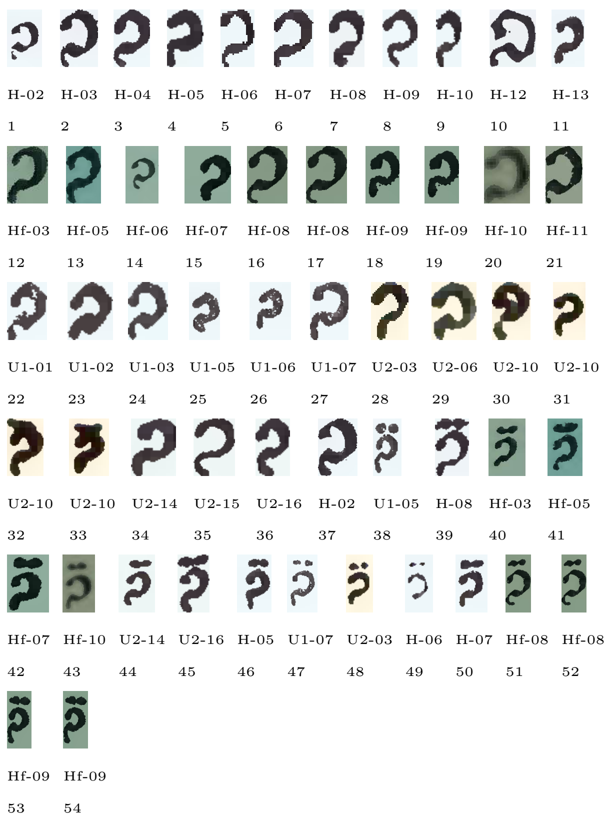

H-02 1
H-03 2
H-04 3
H-05 4
H-06 5
H-07 6
H-08 7
H-09 8
H-10 9
H-12 10
H-13 11
Hf-03 12
Hf-05 13
Hf-06 14
Hf-07 15
Hf-08 16
Hf-08 17
Hf-09 18
Hf-09 19
Hf-10 20
Hf-11 21
U1-01 22
U1-02 23
U1-03 24
U1-05 25
U1-06 26
U1-07 27
U2-03 28
U2-06 29
U2-10 30
U2-10 31
U2-10 32
U2-10 33
U2-14 34
U2-15 35
U2-16 36
H-02 37
U1-05 38
H-08 39
Hf-03 40
Hf-05 41
Hf-07 42
Hf-10 43
U2-14 44
U2-16 45
H-05 46
U1-07 47
U2-03 48
H-06 49
H-07 50
Hf-08 51
Hf-08 52
Hf-09 53
Hf-09 54

Figure 16: The letter small *con*

LATIN SMALL LETTER REVERSED C with representative glyph ‘ɔ’. As the name suggests, it meaning is just *con* (perhaps with some exceptions).

As for the letter with diacritics, the situation is much more complicated. I have not yet found their occurrences in the texts, so I don’t know their meaning. Another problem is the form of the diacritics. Besides a diaresis, we have the diacritic which seems to be the same as the one described in [10] as *jagged horizontal line* which is encoded in Unicode as U+1DD3 COMBINING LATIN SMALL LETTER FLATTENED OPEN A ABOVE but rendered differently: in Junicode it is ‘◌ᷓ’ and in Symbola it is ◌ᷓ. Moreover there is an open question what kind of diacritics, if any, are used in Ungler’s font 10 (items 32 and 33).

4 Modifications of the letter *b*

Old texts used many variations of the letter *b*, many of which are assigned code points by MUFI. Many variants of the letter *b* are also listed in *Polonia Typographica*. Fig. 17 presents those instances which are directly relevant to our purposes here.

We will focus on item 3 (Haller’s font no 4) and those from Hochfeder’s fonts, items 8–17, as their shapes seem to be carefully designed while other

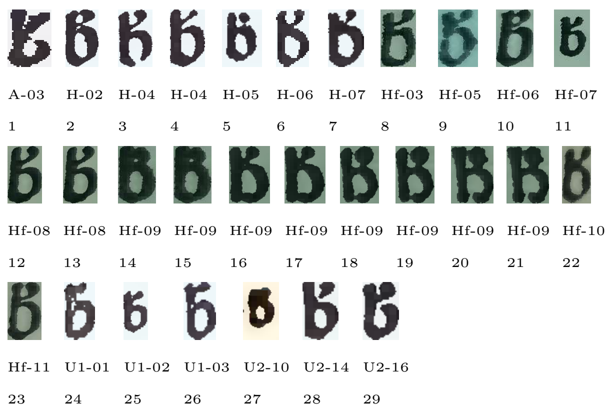

A-03 1
H-02 2
H-04 3
H-04 4
H-05 5
H-06 6
H-07 7
Hf-03 8
Hf-05 9
Hf-06 10
Hf-07 11
Hf-08 12
Hf-08 13
Hf-09 14
Hf-09 15
Hf-09 16
Hf-09 17
Hf-09 18
Hf-09 19
Hf-09 20
Hf-09 21
Hf-10 22
Hf-11 23
U1-01 24
U1-02 25
U1-03 26
U2-10 27
U2-14 28
U2-16 29

Figure 17: The variants of the letter *b*

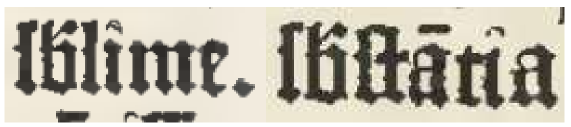


Figure 18: Modified letter *b* in Gutenberg’s bible: *sublime* and *substantia* (Bodleian Library copy, page 21 and 64 of volume II)

items seem to be just more or less crude variations of those.

A character with an almost identical shape appeared already in Gutenberg’s 42-line bible. Despite this, it seems it still has no name and even no generally accepted description. In [2, p. 12] Jacques André proposes the name *latin small letter b with flourish* (he considers also an alternative *latin small letter b ligated with arm of latin small r*, but cf. sec. 8).

According to Gerald Bettridge [4], it means *bis* and, after the long *s*, *ub* (see Fig. 18, also [2, p. 12] and [3, p. 12]). It can be ligated with long *s*; see sec. 13.

It seems this was not always a brevigraph, sometimes it is just equivalent to a normal *b* [6, p. 8]. Or perhaps it was just a printer’s mistake?

5 Modifications of the letter *d*

The similarity of items 6 and (e.g.) 19 to item 3 from Fig. 17 and items 5 and (e.g.) 8 from Fig. 22, all from respectively the same fonts, seems to be a design decision.

I think this is the brevigraph called *d with two ascenders* by Erin Blake in [10]; she states that the brevigraph stands for

> *de* and (depending on the language) also for *der*, *dis*, *dum* and other *d*-syllables

Peter Baker suggested[19] treating the character as MUFI U+F159 LATIN ABBREVIATION SIGN SMALL

[19] `github.com/psb1558/Junicode-font/discussions/133`

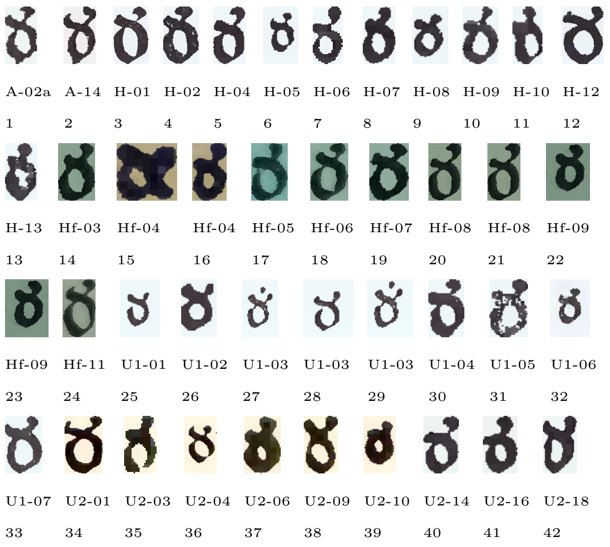

A-02a A-14 H-01 H-02 H-04 H-05 H-06 H-07 H-08 H-09 H-10 H-12
1 2 3 4 5 6 7 8 9 10 11 12
H-13 Hf-03 Hf-04 Hf-04 Hf-05 Hf-06 Hf-07 Hf-08 Hf-08 Hf-09
13 14 15 16 17 18 19 20 21 22
Hf-09 Hf-11 U1-01 U1-02 U1-03 U1-03 U1-03 U1-04 U1-05 U1-06
23 24 25 26 27 28 29 30 31 32
U1-07 U2-01 U2-03 U2-04 U2-06 U2-09 U2-10 U2-14 U2-16 U2-18
33 34 35 36 37 38 39 40 41 42

Figure 19: The variants of the letter *d*

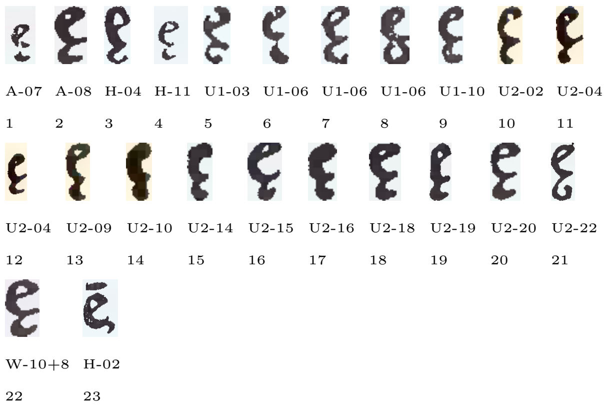

A-07 A-08 H-04 H-11 U1-03 U1-06 U1-06 U1-06 U1-10 U2-02 U2-04
1 2 3 4 5 6 7 8 9 10 11
U2-04 U2-09 U2-10 U2-14 U2-15 U2-16 U2-18 U2-19 U2-20 U2-22
12 13 14 15 16 17 18 19 20 21
W-10+8 H-02
22 23

Figure 20: The variants of the letter *e*

DE (‘ꝺ̄’), called also LATIN SMALL LETTER D ROTUNDA WITH BAR.[20] He also points to another similar MUFI character, namely U+EBB2 LATIN SMALL LETTER D ROTUNDA WITH ACUTE ‘ꝺ́’.

The meaning of the brevigraph with a dot above is yet to be checked in the texts.

6 Modifications of the letter *e*

Fig. 20 presents the well-known *e caudata*, meaning *ae*.

It is an open controversy whether *e caudata* and the contemporary U+0119 LATIN SMALL LETTER E WITH OGONEK should be considered the same character. Peter Baker wrote[21]

[20] `mufi.info/q.php?p=mufi/chars/unichar/61785`
[21] `junicode.sourceforge.net/ecaudata.html`

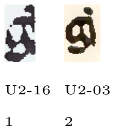


Figure 21: The variants of the letter *g*

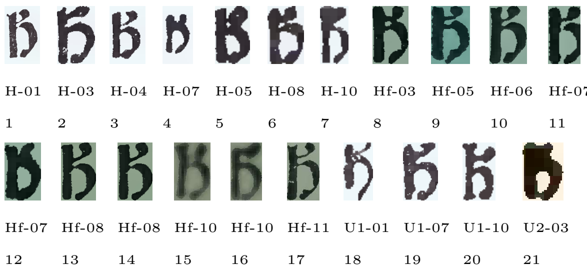

H-01 H-03 H-04 H-07 H-05 H-08 H-10 Hf-03 Hf-05 Hf-06 Hf-07
1 2 3 4 5 6 7 8 9 10 11
Hf-07 Hf-08 Hf-08 Hf-10 Hf-10 Hf-11 U1-01 U1-07 U1-10 U2-03
12 13 14 15 16 17 18 19 20 21

Figure 22: The variants of the letter *h*

> Perhaps it is time to admit that the Latinate cauda and the ogonek used by Polish and other languages are different beasts.

and provided an OpenType feature (*ss15*) to distinguish them in the Junicode font. I have no opinion on this matter.

The meaning of the letter with a bar above (item 19) is yet to be checked in the texts.

7 Modification of the letter *g*

I have little to say about the glyphs in Fig. 21, since I have not found any occurrence of them in a text. On one hand they resemble the letters *rum* (sec. 12), and *tum* (sec. 14). On the other hand it resemble also the glyph from Fig. 2. Moreover, Blake [10] says that *weird vertical line at end of word* means an *s* preceded by a vowel (typically *es* in English and *is* in Latin); in other words in Latin it can be perhaps considered as the ligature of *g* and the letter ‘*is*’ which has been discussed already in sec. 3.

8 Modifications of the letter *h*

For some fonts there is an evident similarity of items among Figures 17, 19, and 22. It seems to be a design decision.

A character with an almost identical shape appeared in Gutenberg's 42-line bible. In the MUFI recommendation, it is identified as U+E8C3 LATIN SMALL LETTER H LIGATED WITH ARM OF LATIN SMALL LETTER R (‘ħ’). Jacques André [2, p. 17] notes that the name is strange and I agree with him.

The glyphs in Fig. 22 seem to be the same as those described as *h with a tick on top* by Blake, who states

> Stands for *h*-syllables like *han*, *het*, and *hic*

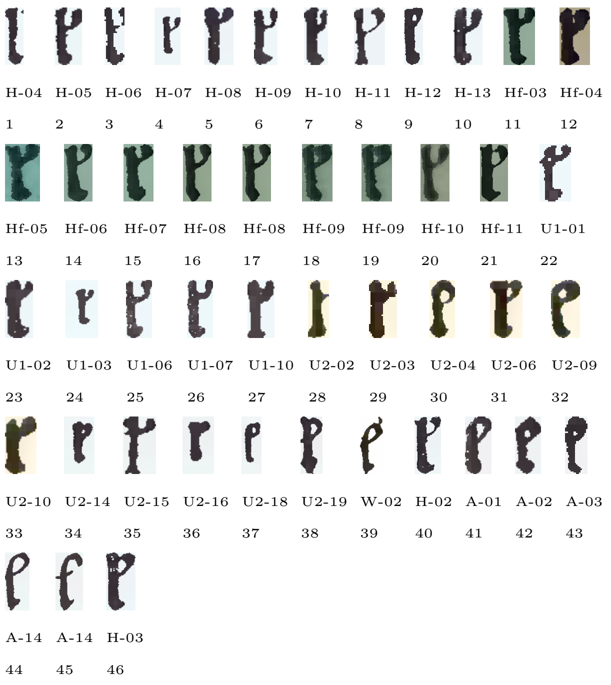

H-04 H-05 H-06 H-07 H-08 H-09 H-10 H-11 H-12 H-13 Hf-03 Hf-04
1 2 3 4 5 6 7 8 9 10 11 12
Hf-05 Hf-06 Hf-07 Hf-08 Hf-08 Hf-09 Hf-09 Hf-10 Hf-11 U1-01
13 14 15 16 17 18 19 20 21 22
U1-02 U1-03 U1-06 U1-07 U1-10 U2-02 U2-03 U2-04 U2-06 U2-09
23 24 25 26 27 28 29 30 31 32
U2-10 U2-14 U2-15 U2-16 U2-18 U2-19 W-02 H-02 A-01 A-02 A-03
33 34 35 36 37 38 39 40 41 42 43
A-14 A-14 H-03
44 45 46

Figure 23: The variants of the letter *l*

Another interpretation was presented by Lisa Howarth on the Facebook The Paleography Society group:[22]

> When attached to an ‘h’, it usually means ‘er’ or ‘ab’ depending on the word

She considers the glyph to be composed from the letter *h* and an diacritical sign, similar to Peter Baker, who identifies[23] the diacritics as U+0335 COMBINING SHORT STROKE OVERLAY (‘◌̵’).

9 The modification of the letter *l*

Reportedly the glyphs in Fig. 23 have the same meaning as U+A749 LATIN SMALL LETTER L WITH HIGH STROKE (‘ꝉ’) and therefore a separate code point has not been assigned.[24] However the Junicode font contains at code point U+F000F the glyph *l with high stroke ending with flourish* (‘ꝉ’), accessible also as *l* with the OpenType feature ss10 and the tags ‘S’‘ꝉ’.

A character with an almost identical shape appeared in Gutenberg’s 42-line bible; cf. Fig. 24.

10 Modifications of the letter *p*

Fig. 25 contains the variants of a well-known brevigraph, available in Unicode since version 5.1.0 (published in 2008) as U+A751 LATIN LETTER P WITH

[22] facebook.com/groups/7687162686/permalink/10158299890607687
[23] github.com/psb1558/Junicode-font/discussions/134
[24] github.com/psb1558/Junicode-font/issues/4

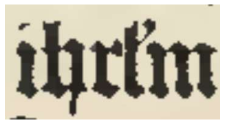


Figure 24: Modified letter *l* in Gutenberg’s bible: according to [4] *iherusalem* (Bodleian Library copy, page 574 of volume II)

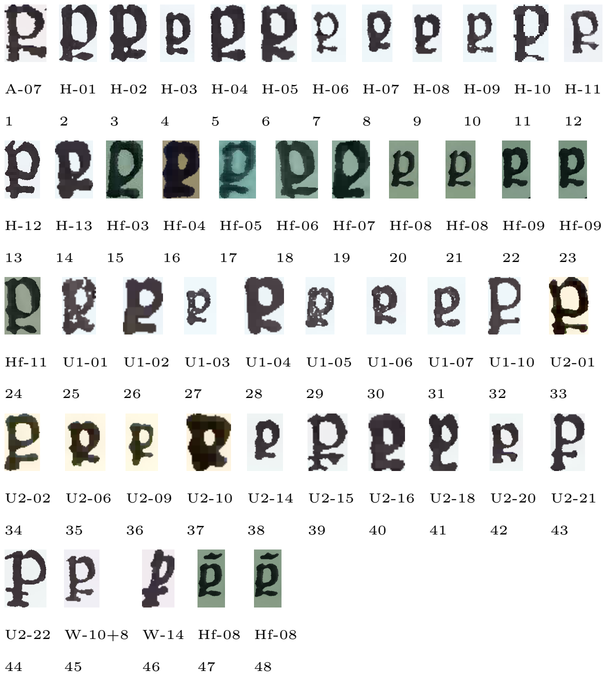

A-07 H-01 H-02 H-03 H-04 H-05 H-06 H-07 H-08 H-09 H-10 H-11
1 2 3 4 5 6 7 8 9 10 11 12
H-12 H-13 Hf-03 Hf-04 Hf-05 Hf-06 Hf-07 Hf-08 Hf-08 Hf-09 Hf-09
13 14 15 16 17 18 19 20 21 22 23
Hf-11 U1-01 U1-02 U1-03 U1-04 U1-05 U1-06 U1-07 U1-10 U2-01
24 25 26 27 28 29 30 31 32 33
U2-02 U2-06 U2-09 U2-10 U2-14 U2-15 U2-16 U2-18 U2-20 U2-21
34 35 36 37 38 39 40 41 42 43
U2-22 W-10+8 W-14 Hf-08 Hf-08
44 45 46 47 48

Figure 25: The letter *p* with stroke

STROKE THROUGH DESCENDER with representative glyph ‘ꝑ’. The brevigraph is ambiguous; the most popular meanings are *per*, *par* and *por*. It can be used as an individual word or as a prefix.

The base characters in Fig. 26 are also the variants of a well-known brevigraph, available in Unicode since version 5.1.0 (published in 2008) as U+A753 LATIN LETTER P WITH FLOURISH with representative glyph ‘ꝓ’. The brevigraph is ambiguous; the most popular meanings are *pro* and *por*. It too can be used as an individual word or as a prefix.

The last characters are included in the MUFI recommendation as U+EED7 LATIN SMALL LIGATURE PP WITH FLOURISH with the glyph ‘ꝓp’; the meaning is *prop-*.

The meaning of the characters with diacritics is yet to be checked in the texts.

11 Modifications of the letter *q*

The glyphs in Fig. 27 represent a well-known brevigraph, included in Unicode since version 5.1.0 (published in 2008) as U+A757 LATIN SMALL LETTER Q

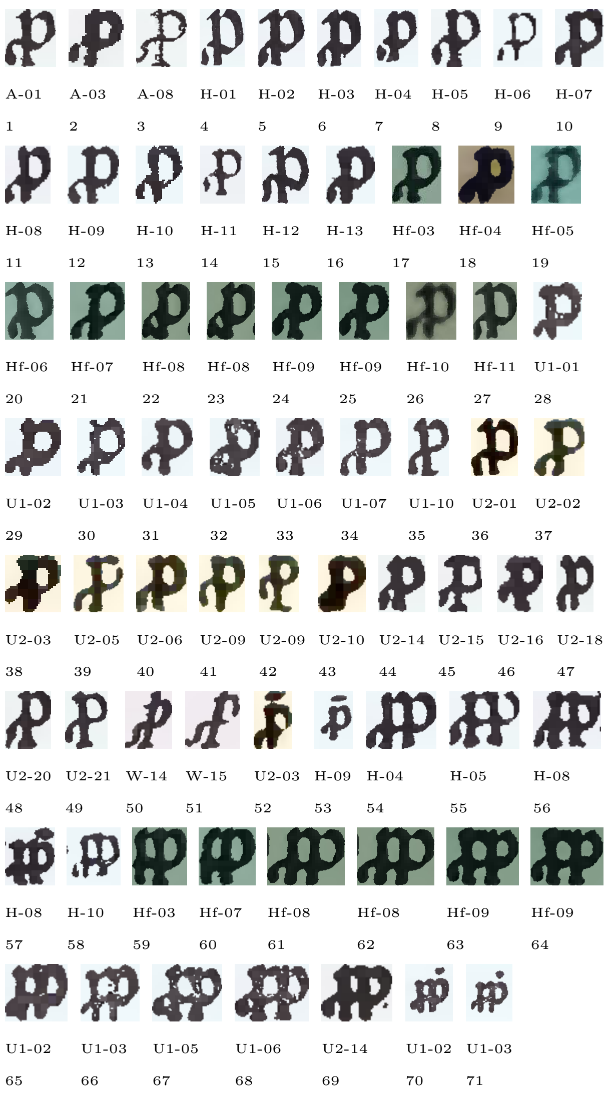

A-01 A-03 A-08 H-01 H-02 H-03 H-04 H-05 H-06 H-07
1 2 3 4 5 6 7 8 9 10
H-08 H-09 H-10 H-11 H-12 H-13 Hf-03 Hf-04 Hf-05
11 12 13 14 15 16 17 18 19
Hf-06 Hf-07 Hf-08 Hf-08 Hf-09 Hf-09 Hf-10 Hf-11 U1-01
20 21 22 23 24 25 26 27 28
U1-02 U1-03 U1-04 U1-05 U1-06 U1-07 U1-10 U2-01 U2-02
29 30 31 32 33 34 35 36 37
U2-03 U2-05 U2-06 U2-09 U2-09 U2-10 U2-14 U2-15 U2-16 U2-18
38 39 40 41 42 43 44 45 46 47
U2-20 U2-21 W-14 W-15 U2-03 H-09 H-04 H-05 H-08
48 49 50 51 52 53 54 55 56
H-08 H-10 Hf-03 Hf-07 Hf-08 Hf-08 Hf-09 Hf-09
57 58 59 60 61 62 63 64
U1-02 U1-03 U1-05 U1-06 U2-14 U1-02 U1-03
65 66 67 68 69 70 71

Figure 26: The letter *p* with flourish

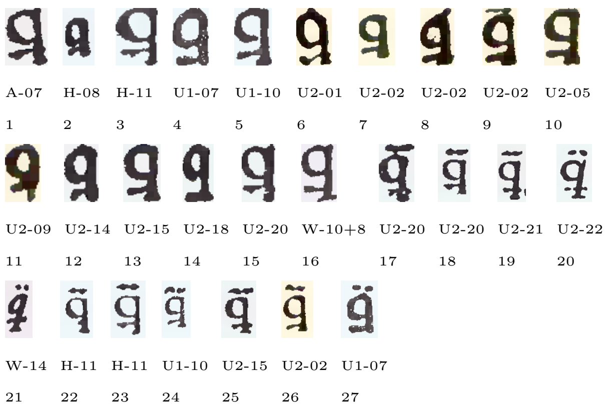

A-07 H-08 H-11 U1-07 U1-10 U2-01 U2-02 U2-02 U2-02 U2-05
1 2 3 4 5 6 7 8 9 10
U2-09 U2-14 U2-15 U2-18 U2-20 W-10+8 U2-20 U2-20 U2-21 U2-22
11 12 13 14 15 16 17 18 19 20
W-14 H-11 H-11 U1-10 U2-15 U2-02 U1-07
21 22 23 24 25 26 27

Figure 27: The letter *q* with stroke through descender

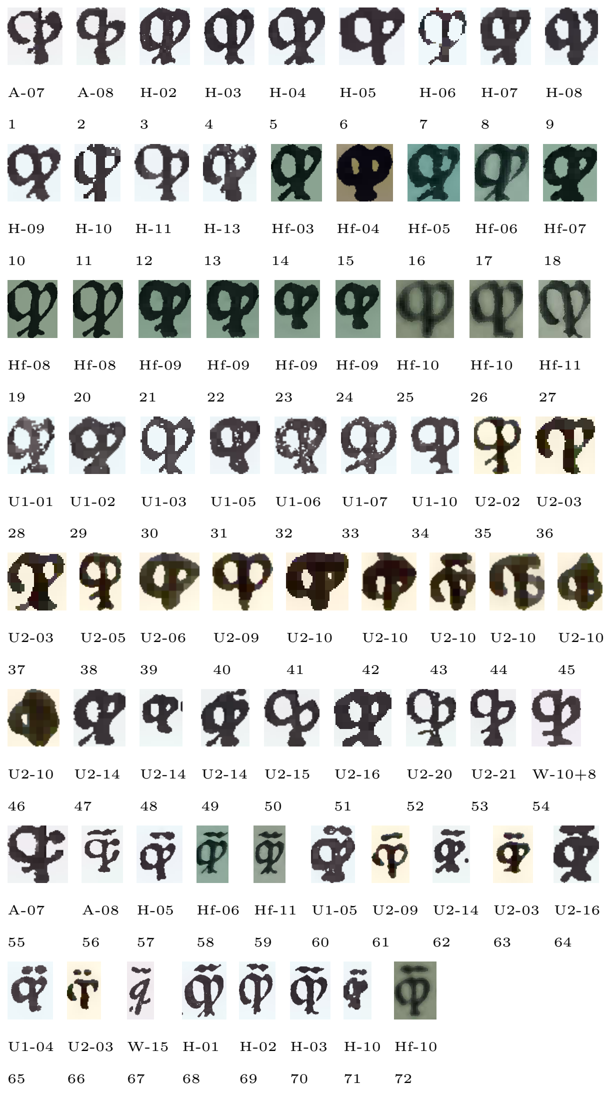

A-07 A-08 H-02 H-03 H-04 H-05 H-06 H-07 H-08
1 2 3 4 5 6 7 8 9
H-09 H-10 H-11 H-13 Hf-03 Hf-04 Hf-05 Hf-06 Hf-07
10 11 12 13 14 15 16 17 18
Hf-08 Hf-08 Hf-09 Hf-09 Hf-09 Hf-09 Hf-10 Hf-10 Hf-11
19 20 21 22 23 24 25 26 27
U1-01 U1-02 U1-03 U1-05 U1-06 U1-07 U1-10 U2-02 U2-03
28 29 30 31 32 33 34 35 36
U2-03 U2-05 U2-06 U2-09 U2-10 U2-10 U2-10 U2-10 U2-10
37 38 39 40 41 42 43 44 45
U2-10 U2-14 U2-14 U2-14 U2-15 U2-16 U2-20 U2-21 W-10+8
46 47 48 49 50 51 52 53 54
A-07 A-08 H-05 Hf-06 Hf-11 U1-05 U2-09 U2-14 U2-03 U2-16
55 56 57 58 59 60 61 62 63 64
U1-04 U2-03 W-15 H-01 H-02 H-03 H-10 Hf-10
65 66 67 68 69 70 71 72

Figure 28: The letter *q* with diagonal stroke

WITH STROKE THROUGH DESCENDER with representative glyph ‘ꝗ’. It can be used as an individual word or as part of it and is quite ambiguous; the reported meanings are *quam*, *que*, *quan-* and *qui-*.

Fig. 27 demonstrates also various kinds of diacritical marks which can be used with this brevigraph. The meaning of modified brevigraphs is not clear and requires further research.

The characters in Fig. 28 are in my opinion variants of the brevigraph introduced to Unicode in version 5.1.0 (published 2008) as U+A759 LATIN SMALL LETTER Q WITH DIAGONAL STROKE with representative glyph ‘ꝙ’, although such a classification of some shapes is questionable. The Unicode name is not very adequate; in [3, p. 70] the name LATIN SMALL LETTER Q WITH SWASH is proposed. The brevigraph has three meanings: *quod*, *qui* and *que*; it can be used as an individual word or as a part of it.

The meaning of modified brevigraphs with diacritical marks is not clear. Here we will mention only that in [3, p. 71] a glyph similar to those from Fig. 28 is classified as LATIN SMALL LETTER Q WITH SWASH AND LATIN SMALL LETTER FLATTENED OPEN A ABOVE (an alternative name LATIN SMALL LETTER Q WITH FLOURISH ... is also considered), and an example is given where the brevigraph means *quan-*.

The glyphs in Fig. 29 represent the brevigraph assigned the Private Use Area code U+E8BF and the name LATIN SMALL LETTER Q LIGATED WITH FINAL ET by MUFI in version 4 of the recommendation [11, p. 81]. In the Junicode font, it is rendered as ‘’.

Finding the meaning of the brevigraph with a diacritical mark requires additional research, but we will note that according to [10] some of the glyphs from Fig. 29 mean *quam* or *quan*.

12 Modifications of the letter *r*

The first glyph in Fig. 30 is an interesting and rather little-known character. Although this is far from obvious, it is U+A775 LATIN SMALL LETTER RUM despite the fact that the Unicode representative glyph is ‘ꝵ’, as in Junicode we have ‘ꝵ’ — the glyph is practically identical to that on the figure. The character was added to Unicode in version 5.1 (published in 2008), along with some similar characters (see sec. 14). I assume the second glyph in the figure is just a variant of the first one.

The glyphs in Fig. 31 are ambiguous. They can represent U+A776 LATIN LETTER SMALL CAPITAL RUM (ꝶ), but they can also be interpreted as U+211E PRESCRIPTION TAKE (‘℞’) and, last but not least, U+211F RESPONSE (‘℟’) which in prayer books can be paired with the *versicle* character (see sec. 15).

13 Modifications of the letter long *s*

The glyphs in Fig. 32 are noted in the MUFI recommendations as M+E8B7 LATIN SMALL LETTER LONG S WITH FLOURISH (‘’).

The glyphs in Fig. 33 are present neither in Unicode nor in the MUFI recommendation, but they are obviously the long *s* (U+017F) ligated with the final *et* (U+A76B). As far as I know, this ligature is available only in the Junicode font[25] with the Historic Ligature (*hlig*) feature: ‘ſꝫ’. The meaning is *sed*, as exemplified in [9, example (68)].

Fig. 34 shows a problematic glyph which I'm not sure how to interpret.

One of the component characters of the ligatures presented in Fig. 35 has been already mentioned in

[25] github.com/psb1558/Junicode-font/discussions/140

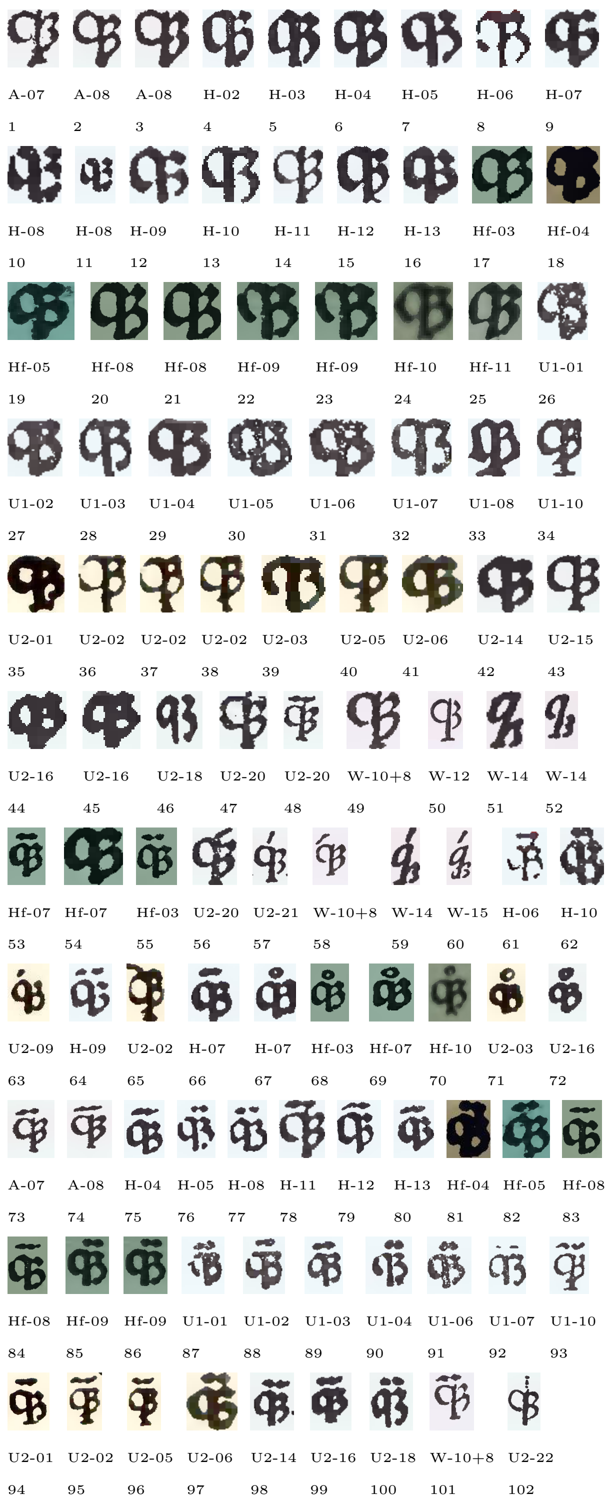

A-07 1 A-08 2 A-08 3 H-02 4 H-03 5 H-04 6 H-05 7 H-06 8 H-07 9
H-08 10 H-08 11 H-09 12 H-10 13 H-11 14 H-12 15 H-13 16 Hf-03 17 Hf-04 18
Hf-05 19 Hf-08 20 Hf-08 21 Hf-09 22 Hf-09 23 Hf-10 24 Hf-11 25 U1-01 26
U1-02 27 U1-03 28 U1-04 29 U1-05 30 U1-06 31 U1-07 32 U1-08 33 U1-10 34
U2-01 35 U2-02 36 U2-02 37 U2-02 38 U2-03 39 U2-05 40 U2-06 41 U2-14 42 U2-15 43
U2-16 44 U2-16 45 U2-18 46 U2-20 47 U2-20 48 W-10+8 49 W-12 50 W-14 51 W-14 52
Hf-07 53 Hf-07 54 Hf-03 55 U2-20 56 U2-21 57 W-10+8 58 W-14 59 W-15 60 H-06 61 H-10 62
U2-09 63 H-09 64 U2-02 65 H-07 66 H-07 67 Hf-03 68 Hf-07 69 Hf-10 70 U2-03 71 U2-16 72
A-07 73 A-08 74 H-04 75 H-05 76 H-08 77 H-11 78 H-12 79 H-13 80 Hf-04 81 Hf-05 82 Hf-08 83
Hf-08 84 Hf-09 85 Hf-09 86 U1-01 87 U1-02 88 U1-03 89 U1-04 90 U1-06 91 U1-07 92 U1-10 93
U2-01 94 U2-02 95 U2-05 96 U2-06 97 U2-14 98 U2-16 99 U2-18 100 W-10+8 101 U2-22 102

Figure 29: The letter *q* with final *et*

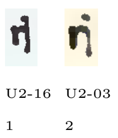


Figure 30: The alternative glyphs of the letter *rum*

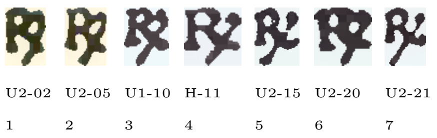
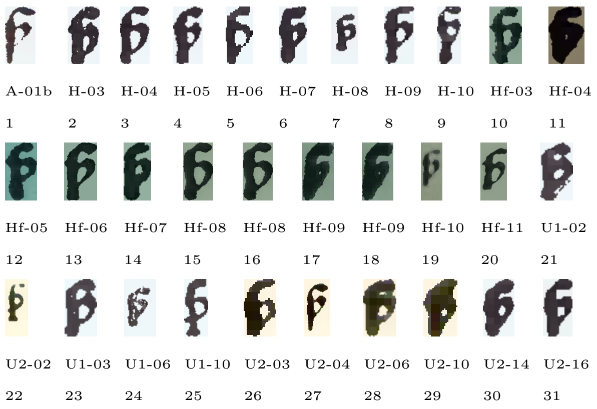
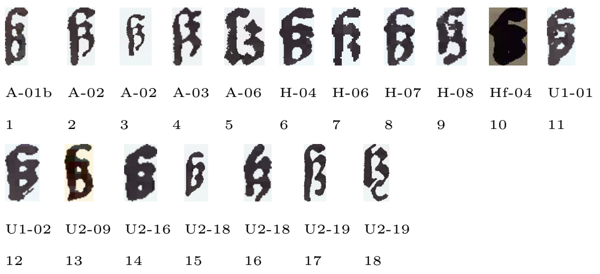


Figure 31: The alternative glyphs of the letter *response*


A-01b H-03 H-04 H-05 H-06 H-07 H-08 H-09 H-10 Hf-03 Hf-04
1 2 3 4 5 6 7 8 9 10 11
Hf-05 Hf-06 Hf-07 Hf-08 Hf-08 Hf-09 Hf-09 Hf-10 Hf-11 U1-02
12 13 14 15 16 17 18 19 20 21
U2-02 U1-03 U1-06 U1-10 U2-03 U2-04 U2-06 U2-10 U2-14 U2-16
22 23 24 25 26 27 28 29 30 31
Figure 32: Long *s* with flourish


A-01b A-02 A-02 A-03 A-06 H-04 H-06 H-07 H-08 Hf-04 U1-01
1 2 3 4 5 6 7 8 9 10 11
U1-02 U2-09 U2-16 U2-18 U2-18 U2-19 U2-19
12 13 14 15 16 17 18
Figure 33: Long *s* with final *et*

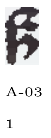
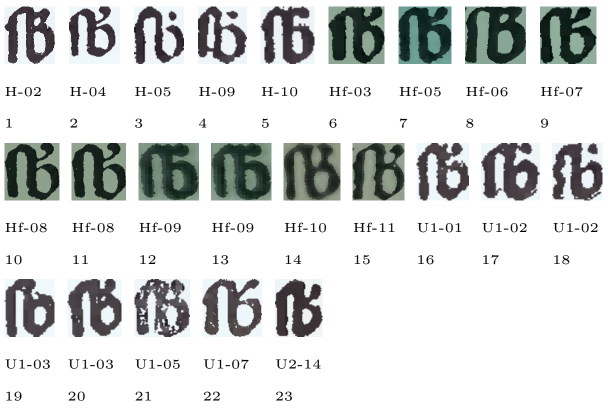


Figure 34: Long *s* with final *et*?


H-02 H-04 H-05 H-09 H-10 Hf-03 Hf-05 Hf-06 Hf-07
1 2 3 4 5 6 7 8 9
Hf-08 Hf-08 Hf-09 Hf-09 Hf-10 Hf-11 U1-01 U1-02 U1-02
10 11 12 13 14 15 16 17 18
U1-03 U1-03 U1-05 U1-07 U2-14
19 20 21 22 23
Figure 35: The ligature of long *s* with the letter *b* and its modifications

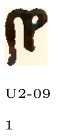


Figure 36: The ligature of long *s* with the letter *l* with flourish?

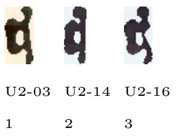


Figure 37: The letter *tum*

sec. 4. We see also the letter *b* with a dot above; the meaning of the letter, ligated or not, is yet to be investigated.

It is worth noting that in item 18 instead of a normal long *s* we have a LONG FUNNY S proposed to be included in the MUFI recommendation.[26]

In Fig. 36 we have a ligature which can be perhaps treated as a variant of MUFI U+E8AF LATIN SMALL LIGATURE LONG S L WITH DIAGONAL STROKE (‘’).

14 Modifications of the letter *t*

Fig. 37 shows the rather rare brevigraph U+A777 LATIN SMALL LETTER TUM with representative glyph ‘ꝷ’. It was introduced in version 5.1 (published in 2008), together with some related letters such as U+A775 LATIN SMALL LETTER RUM (see sec. 12).

15 Modification of the letter v

The primary interpretation of the glyphs in Fig. 38 seems to be U+A75F LATIN SMALL LETTER V WITH

[26] `mufi.info/q.php?p=mufi/chars/unichar/1048876`

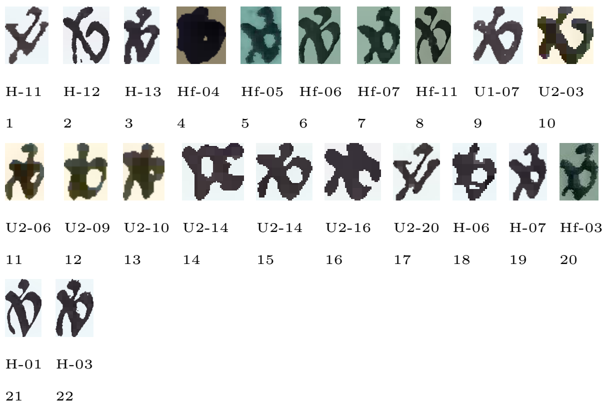


Figure 38: The letter *v* with diagonal stroke

DIAGONAL STROKE ('ꝟ') added to Unicode in version 5.1.0 (published in 2008); in item 14 it is the other arm which is crossed. It means *ver* or *vir*.

The glyphs can stand also for U+2123 VERSICLE ('℣'), used to mark in the prayer books the beginning of a versicle, i.e., a short sentence said or sung by the minister in a church service, to which the congregation gives a response.[27]

16 Final remarks

As has already been mentioned, the next step should be to find the occurrences of the discussed brevigraphs in the texts and in this way find or verify their meaning. For this, we don't need the full transcriptions of the texts. What is sufficient for our purposes is *glyph* or *character spotting*. These tasks are discussed in some publications, but there seems to be no tool available directly for use. With some limitations, a variant of a workflow described earlier can be used for this purpose. The djvudict output can be converted to a PDF document (created with TEX) with enlarged glyphs which make it relatively easy to search for interesting items and to note their identifiers (Fig. 39). Additionally, a djview4poliqarp index can be created, which uses the shape identifier as the searchable entry fields (Fig. 40). The identifiers are not unique, nevertheless it is possible with some effort to find the context of a shape in the document, as illustrated in Figures 39 and 40 (note the shape 01344).

At present my programs supporting this approach are too primitive to be used conveniently, but I will try to improve them. Any help from Python and/or QT programmers (QT was used to implement djview4poliqarp and djview4shapes) would be welcomed and appreciated.

[27] Definition provided by Google in a non-linkable form.

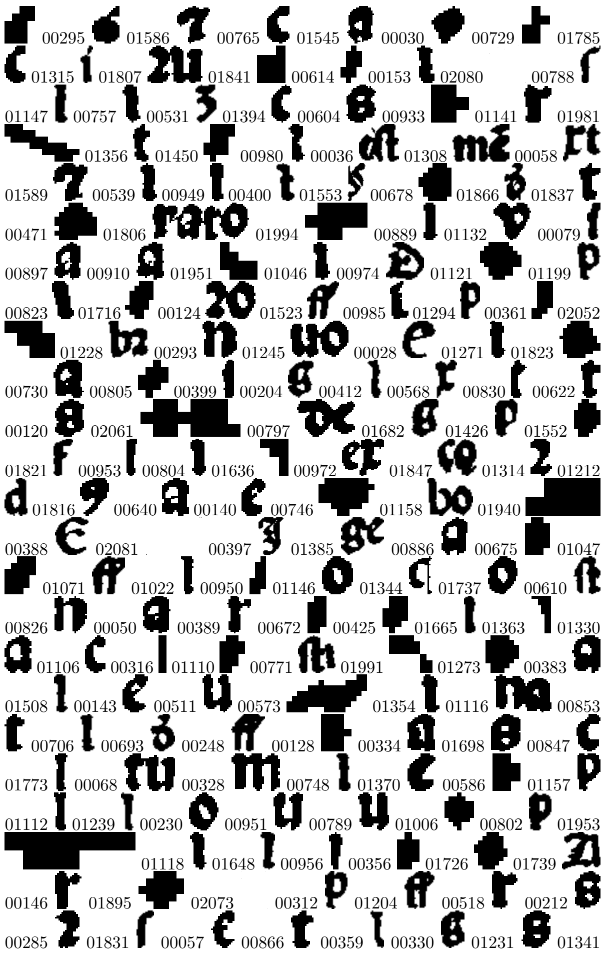
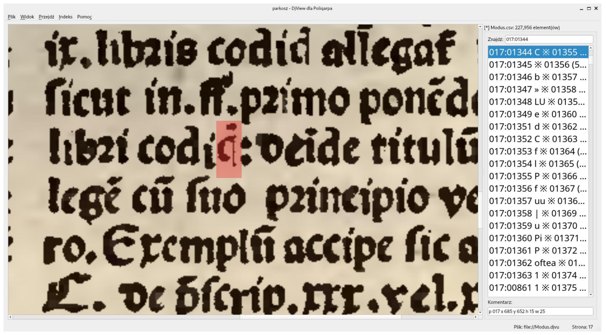


Figure 39: The djvudict output in the form of a PDF document


parkosz - DjView dla Poliqarpa
Plik Widok Przejdź Indeks Pomoc

[*] Modus.csv: 227,956 element(ów)

Znajdź: 017:01344

017:01344 C ※ 01355 ...
017:01345 ※ 01356 (5...
017:01346 b ※ 01357 ...
017:01347 » ※ 01358 ...
017:01348 LU ※ 0135...
017:01349 e ※ 01360 ...
017:01351 d ※ 01362 ...
017:01352 C ※ 01363 ...
017:01353 f ※ 01364 (...
017:01354 l ※ 01365 (...
017:01355 P ※ 01366 ...
017:01356 f ※ 01367 (...
017:01357 uu ※ 0136...
017:01358 | ※ 01369 ...
017:01359 u ※ 01370 ...
017:01360 Pi ※ 01371...
017:01361 P ※ 01372 ...
017:01362 oftea ※ 01...
017:01363 1 ※ 01374 ...
017:00861 1 ※ 01375 ...

Komentarz: p 017 x 685 y 652 h 15 w 25

Plik: file://Modus.djvu Strona: 17

Figure 40: The djvudict output in the form of a djview4poliqarp index

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, R. Jimenes. Transcription et codage des imprimés de la Renaissance. *Revue des Sciences et Technologies de l'Information —*

Série Document Numérique, 16(3):113–139, 2013. doi.org/10.3166/DN.16.3.113-139

[2] 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

[3] 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

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

[5] J.S. Bień. 16th century Latin brevigraphs in Unicode — a computer resource. In *Grapholinguistics in the 21st Century 2022. Proceedings. Grapholinguistics and Its Applications*, Y. Haralambous, ed., pp. 31–46. Fluxus Editions, Brest, 2023 (to appear). doi.org/10.36824/2022-graf-bien

[6] 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

[7] J.S. Bień. Representing Parkosz's alphabet in the Junicode font. *TUGboat* 43(3):247–251, 2022. doi.org/10.47397/tb/43-3/tb135bien-parkosz

[8] 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

[9] J.S. Bień. Towards an inventory of old print characters: Ungler's *Rubricella*, a case study – errata. *TUGboat* 45(1):44, 2024. doi.org/10.47397/tb/45-1/tb139bien-rubricella-errata

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

[11] O.E. Haugen. *MUFI character recommendation version 4.0*. Medieval Unicode Font Initiative, 2015. hdl.handle.net/1956/10699

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

Towards an inventory of old print characters: Ungler's *Rubricella*, a case study — Errata

Janusz S. Bień

Abstract

Errata for the article in *TUGboat* issue 44:3, pp. 364–375 (tug.org/TUGboat/tb44-3/tb138bien-rubricella.pdf): 1) ‘‘Ŋ’ and ‘ꟼ’ were typeset where ‘ꝛ’ and ‘lʼ’ were intended; 2) ‘’ was confused with ‘ſꝫ’; 3) an *h* should have been *b*.

4.5 Brevigraphs

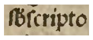
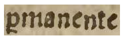
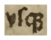
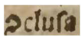
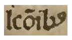
From the first paragraph of this section in the original article:

> ... in Junicode also “‘Ŋ” ...
> ... in the Junicode font also “ꟼ“ ...

The wrong characters were typeset; these should have been ‘ꝛ’ and ‘lʼ’. This was because of a problem with the default font renderer in Lua(LA)TEX. Switching to the HarfBuzz renderer solves it. Thanks to Marcel Krueger and Luigi Scarso.

A second error was towards the end of the section. The figure (below, corrected from the original) lists two brevigraphs based on the long *s*. The first one in example (65) is ‘’ (M+E8B7 LATIN SMALL LETTER LONG S WITH FLOURISH [MUFI 4.0]). The second, in example (68), is similar and in the paper was confused with it. Actually it is the long *s* (U+017F) ligated with the final *et* (U+A76B); it is present neither in Unicode nor in the MUFI recommendation. As far as I know, this ligature is available only in the Junicode font[1] with the Historic Ligature (*hlig*) feature: ‘ſꝫ’. It can be seen in example (68) where it is used as a separate word (its interpretations were suggested in the Facebook Paleography Society group by Gionata Brusa and Carolus Hrachowiczensis;[2] unfortunately neither I nor none of the group noted that I had confused ‘’ with ‘ſꝫ’).

The first brevigraph, example (63) with the ligature with the letter *b* (the original article incorrectly wrote *h* here) and a diacritic mark, was discussed above.

					
(63)	(64)	(65)	(66)	(67)	(68)
ſ̄bſcripto	ꝑmanente	ꝙ	vſqꝫ	q̈ꝫ	ſꝫ
subscripto	permanente	quod	usque	quam	scilitet? sed?
					
(69)	(70)				
ꝯcluſa	lcōibꝰ				
conclusa	lectionibus				

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

[1] github.com/psb1558/Junicode-font/discussions/140
[2] facebook.com/groups/7687162686/posts/10159250228377687

doi.org/10.47397/tb/45-1/tb139bien-rubricella-errata

The DuckBoat — Beginners' Pond: Tcolorchat!

Herr Professor Paulinho van Duck

Abstract

In this installment, Prof. van Duck will show you how to typeset chat-like conversations using `tcolorbox`, a package for creating customized text boxes.

1 Wunderschöne Tage in Bonn!

Hi, (LA)TEX friends!

Last July, for the first time, my assistant Carla and I attended a TUG meeting in person. We spent some exciting days in Bonn, Germany, listening to captivating talks, in an astonishing location, a former monastery.

We met a lot of friends from countries all around the world, Europe, of course, but also the USA, Brazil, and India! We joyfully gave a face to many nicknames.

Our local guest, Gert Fischer, took us on a fascinating city tour. We found out interesting things about Bonn, such as that the firm famous for inventing gummi bears and licorice wheels is based there! We thank him, Ulrike Fischer, and the other members of the conference committee for the perfect organization.

We dined at a skyscraper restaurant with a gorgeous view of the Rhine Valley. We took a river trip to Königswinter, enjoyed the view from the hill, and visited the ruins of Drachenfels Castle. We also stopped at Schloss Drachenburg, technically not a castle, but a nineteenth-century mansion.

We had a lot of fun and, last but not least, David Carlisle did not eat me,[1] quack!

Talking with some TEXnicians at the meeting, I found out that the `tabularray` package has a flaw I should have mentioned in my previous article [4].

Unfortunately, `tabularray` significantly slows down the compilation of your document. If you have more than a dozen tables (the exact number depends on their complexity), it is better to use the traditional environments, possibly reserving `tabularray` for the few out-of-the-ordinary tables.

It is a pity that such a gorgeous package has this defect, I hope it will be somehow fixed in the future.

[1] For the ones who do not usually attend TEX.SE chat: see Box 1 and footnote 3.

In this installment, I would like to show you some features of `tcolorbox`, a convenient package to typeset colored and framed text boxes. The idea comes from the post *Chat bubbles with picture and name attached*[2] on TEX.SE.

The dialogues in the examples (except for Box 1) are taken from famous movies. Try to guess them without searching on the internet, quack!

Eventually, I would like to thank the user *frougon* from TopAnswers TeX (`topanswers.xyz/tex`) for their precious help with the examples in Boxes 5 and 6; as well as samcarter, Gert Fischer, and Enrico Gregorio for their proofreading and suggestions.

2 Quack Guide n. 9: Chat-like messages with `tcolorbox`

The package `tcolorbox` is very versatile, allowing creation of text boxes with a lot of features. It also has a detailed manual [6], rich in useful examples. I strongly advise you to look at its Section 2 *Quick Reference*, where some of the main options are visualized.

For the sake of brevity, I will not illustrate all the possible customizations you can apply to your boxes. I will only try to create something that resembles a chat dialogue, proceeding step by step, adding a few options at a time to get the desired result.

The main environment of the package is

```
\begin{tcolorbox}[⟨options⟩]
⟨environment content⟩
\end{tcolorbox}
```

In the optional argument, you can list all the specific ⟨*options*⟩ for that box.

The ⟨*environment content*⟩ can be divided into a mandatory upper part and an optional lower part. In this installment, I will use `tcolorbox`es with the upper part only.

There is also a command

```
\tcbox[⟨options⟩]{⟨box content⟩}
```

that creates a colored box fitted to the width of ⟨*box content*⟩. The main differences from the `tcolorbox` environment are that it cannot have a lower part and cannot be broken across pages.

In some of the following examples, you will only find the `tcolorbox` environments and the possible settings needed for them.

Remember to first load the package in your preamble:

```
\usepackage{tcolorbox}
```

[2] `tex.stackexchange.com/questions/624775/chat-bubbles-with-picture-and-name-attached`

doi.org/10.47397/tb/45-1/tb139duck-tcolorchat

2.1 The standard

Standard `tcolorbox`es appear as black and white boxes with rounded corners.

You can also add a heading line with the option `title={`⟨*text*⟩`}`; see Box 1.[3]

Box 1 – Standard `tcolorbox`es

```
\begin{tcolorbox}[title={Paulo}]
  Quack!
\end{tcolorbox}
\begin{tcolorbox}[title=David]
  Dinner!
\end{tcolorbox}
```

Paulo

Quack!

David

Dinner!

The brackets around the title are not mandatory, but it is a good practice to add them, because in some cases, for example, if there is an equal sign or a comma in the title text, they *are* needed.

2.2 Aspect customization and border dimension

Let us start with the title setting, I would like to use it to show who is chatting. The option `fonttitle=`⟨*text*⟩ puts the content of ⟨*text*⟩ before the title. I used it to set a bold, script-size font. I also added a little vertical space before/after the title text with `toptitle`/`bottomtitle=`⟨*length*⟩.

Since I would like to make the box more lively, I added some colors, with `coltitle=`⟨*color*⟩ for the title text, `colbacktitle=`⟨*color*⟩ for the background of the title, and `colback=`⟨*color*⟩ for the background of the main text. See Box 2, and please use your imagination if you are reading this article in black and white.

Since the sent messages should appear horizontally aligned to the right (and the received ones to the left), I took advantage of the options:
`halign title=`⟨*alignment*⟩, for the title (aligned to the left by default), and `halign=`⟨*alignment*⟩, for the content (initially justified).

I also do not like the border lines, so I set their dimension to zero, using `titlerule=`⟨*length*⟩, for the line between the title and the text; `boxrule=`⟨*length*⟩, for the border lines; and `boxsep=`⟨*length*⟩, for the padding between the text content and the frame of the box.

As you can see in Box 2, this is not enough. Even setting their width to zero points, the rules are still visible, even if barely.

Box 2 – Zero dimension rules still visible

```
% In your preamble:
\tcbset{fonttitle=\bfseries\scriptsize,
  titlerule=0pt, boxrule=0pt, boxsep=0pt,
  toptitle=6pt, bottomtitle=2pt}

% In your document:
\begin{tcolorbox}[title={Leia},
  halign=flush left,
  coltitle=red, colbacktitle=yellow!30,
  colback=yellow!30]
  I love you
\end{tcolorbox}
\begin{tcolorbox}[title={Han},
  halign title=flush right,
  halign=flush right,
  coltitle=blue, colbacktitle=green!30,
  colback=green!30]
  I know
\end{tcolorbox}
```

Leia

I love you

Han

I know

Of course, there is a trick to avoid it, quack! The option `opacityframe=`⟨*fraction*⟩ sets the frame opacity of the box to the given fraction. I made it transparent with `opacityframe=0`; see Box 3.

Using `\tcbset{`⟨*options*⟩`}`, you can set options for all top-level `tcolorbox`es in your document. If you would also like to set them for nested boxes, there is `\tcbsetforeverylayer{`⟨*options*⟩`}`.

If you do not want to have a general setting, you can create a `style` and use it only when needed, like `commonoptions` in Box 3. The same Box shows that you can also create customized `tcolorbox` environments, like `sentmsg` and `receivedmsg`, with the command
`\newcolorbox[`⟨*init options*⟩`]{`⟨*name*⟩`}`
  `[`⟨*number*⟩`][`⟨*default*⟩`]{`⟨*options*⟩`}`
where ⟨*name*⟩ is the name of the new environment,

[3] This Box represents the usual conversation between Paulo Cereda and David Carlisle in the TEX.SE chat.

featuring some ⟨*options*⟩, a certain ⟨*number*⟩ of parameters (the first one could be optional with a ⟨*default*⟩ value), and some ⟨*init options*⟩ in case the box should be numbered. See Section 3.2 *Producing* `tcolorbox` *Environments and Commands* of the package documentation [6] for details.

Box 3 – No frame at all

```
% In your preamble:
\tcbset{
  commonoptions/.style={
    fonttitle=\bfseries\scriptsize,
    titlerule=0pt, boxrule=0pt, boxsep=0pt,
    toptitle=6pt, bottomtitle=2pt,
    opacityframe=0,
    }
  }
\newtcolorbox{receivedmsg}[2][]{
  commonoptions,
  halign title=flush left,
  halign=flush left,
  coltitle=red, colbacktitle=yellow!30,
  colback=yellow!30,
  title={#2},
  #1}
\newtcolorbox{sentmsg}[2][]{
  commonoptions,
  halign title=flush right,
  halign=flush right,
  coltitle=blue, colbacktitle=green!30,
  colback=green!30,
  title={#2},
  #1}

% In your document:
\begin{receivedmsg}{Marion}
  You're not the man I knew ten years ago.
\end{receivedmsg}
\begin{sentmsg}{Indiana}
  It's not the years, honey, it's the
    mileage.
\end{sentmsg}
```

Marion
You're not the man I knew ten years ago.

Indiana
It's not the years, honey, it's the mileage.

For more complex environments, you could use `\NewTColorBox`, which is analogous to the macro `\NewDocumentEnvironment`:

`\NewTColorBox[`⟨*init options*⟩`]{`⟨*name*⟩`}`
  `{`⟨*specification*⟩`}{`⟨*options*⟩`}`

The meaning of the parameters is the same as in `\newtcolorbox` but, in addition, you can set the argument ⟨*specification*⟩.

In Boxes 5 and 6, you will find some examples of environments with a starred `s`, an optional `o`, and a mandatory argument `m`. See `usrguide`, the user-mode documentation for LaTeX [2], for a description of all possible argument types.

An alternative method to get rid of the borders is loading the library `skins`:
`\tcbuselibrary{skins}`
and using `skin=enhanced` or just `enhanced`, which translates the drawing commands into Ti*k*Z path commands.

It also allows using the option `frame hidden`, a shortcut for
`frame style={draw=none, fill=none}`
You can find an example in Box 4.

Other than `skins`, `tcolorbox` has several libraries. For example, to cite only some, `listings` or `minted`, which load the homonymous packages for printing source code; `theorems`, which loads `amsmath` and provides macros for typesetting boxed theorems and similar environments; `breakable`, which allows you to automatically break your boxes from one page to another; and `fitting`, which provides support for adapting the font size for your text to the box dimensions.

It is often the case that you need more than one of them. The package makes available three shortcuts to load some or all of them at once: `many`, which loads the libraries for the box settings; `most`, which loads all libraries except `minted` and `documentation`; and `all` that loads all libraries.

For details, see Section 1.3 *Libraries* of the package manual [6].

2.3 Dimension and position

The chat messages are usually positioned on the right or the left, depending on if they are sent or received.

You may create this effect by taking advantage of the options `left/right skip=`⟨*length*⟩; see Box 4. They insert some horizontal space of the given length before/after the `tcolorbox`.

However, this is not enough to mimic the usual look of a chat. If the messages are brief, they do not enlarge to the border of your phone/computer screen. The frame should adapt to the width of the text. You can achieve this using two options together.

Box 4 – Library `skins` and option `skip`

```
% In your preamble:
\usepackage{tcolorbox}
\tcbuselibrary{skins}
\tcbset{
  commonoptions/.style={
    enhanced,
    frame hidden,
    fonttitle=\bfseries\scriptsize,
    titlerule=0pt, boxrule=0pt, boxsep=0pt,
    toptitle=6pt,
    bottomtitle=2pt
    },
  }
\newtcolorbox{receivedmsg}[2][]{
  commonoptions,
  right skip=.7cm,
  halign title=flush left,
  halign=flush left,
  coltitle=red,
  colbacktitle=yellow!30,
  colback=yellow!30,
  title={#2},
  #1}
\newtcolorbox{sentmsg}[2][]{
  commonoptions,
  left skip=.7cm,
  halign title=flush right,
  halign=flush right,
  coltitle=blue,
  colbacktitle=green!30,
  colback=green!30,
  title={#2},
  #1}

% In your document:
\begin{receivedmsg}{Ilsa}
  But what about us?
\end{receivedmsg}
\begin{sentmsg}{Rick}
  We'll always have Paris.
\end{sentmsg}
```

Ilsa
But what about us?

Rick
We'll always have Paris.

The first is `hbox`, a shortcut for `capture=hbox`; it sizes the box according to its content. It is the default for `\tcbox`. It does not allow boxes with a lower part or with text split into more rows.

The latter limitation is the reason why the second option is necessary: `varwidth upper=`⟨*length*⟩. This way, the text of the upper part of the box is put into a `varwidth` environment with a maximal width of ⟨*length*⟩. You need to load the `varwidth` package [1] to use it.

This option also does not permit having a breakable `tcolorbox` or a lower part in the text. The package manual warns that it is only sensible for a `\tcbox`, but it also works in a `tcolorbox` environment with `hbox`.

For further details, see Sections 4.17 *Capture Mode* and 4.11 *Box Content Additions* of the package manual [6].

Pay attention that `varwidth upper` sets the box width to the text width. Thus, if your title is longer than your text, you have to use some tricks.

In Box 5, to set Edna's message text to the width of the box title I used a custom option, `mywidth`, that has the title as a parameter. I took advantage of the possibility of adding some arbitrary code into an option list, using `code=`⟨*code*⟩.

The following passage could be a little demanding for a newbie (and also a bit boring), but if you learn these commands, you will have nice tools for typesetting your documents, quack!

I defined a new length register, using
`\newlength{\`*`lengthname`*`}`
In a register you can store a value and use it for calculations. Since I used it to store the title width, I named it `\mytitlelen`.

I also defined `\mymaxlen` as the length for the maximum width my frame can have, and with
`\setlength{\`*`lengthname`*`}{`⟨*length*⟩`}`
I set this to the ⟨*length*⟩ of `.9\linewidth`.

The command
`\settowidth{\`*`lengthname`*`}{`⟨*some text*⟩`}`
allows setting your length to the width of ⟨*some text*⟩. I used it, in the `code` of the customized option `mywidth`, to set `\mytitlelen` to the width of the title text, appropriately formatted.

Then, with
`\addtolength{\`*`lengthname`*`}{`⟨*length*⟩`}`
I added the left and right space between title text and frame. To get these values, I used the lengths where `tcolorbox` stores them: `\kvtcb@lefttitle` and `\kvtcb@righttitle`. They are not mentioned in the manual, but you can find them if you look at the package code `tcolorbox.sty`. Alert: look but do not touch. Otherwise, you can do damage, quack!

Moreover, since they are internals, you should limit their use to what is strictly necessary (if the package author changed their names, for example, your code would no longer compile).

Box 5 – `hbox` and `varwidth`

```
% In your preamble:
\usepackage[many]{tcolorbox}
\usepackage{varwidth}

\newlength{\mytitlelen}
\newlength{\mymaxlen}
\setlength{\mymaxlen}{.9\linewidth}

\makeatletter
\tcbset{
commonoptions/.style={
  enhanced,
  fonttitle=\bfseries\scriptsize,
  titlerule=0pt, boxrule=0pt, boxsep=0pt,
  toptitle=6pt, bottomtitle=2pt,
  opacityframe=0,
  },
mywidth/.code={%
  \settowidth{\mytitlelen}{%
    \bfseries\scriptsize #1}%
  \addtolength{\mytitlelen}{%
    \kvtcb@lefttitle}%
  \addtolength{\mytitlelen}{%
    \kvtcb@righttitle}%
  \ifdimcomp{\mytitlelen}{>}{\mymaxlen}{%
    \tcbset{width=\mymaxlen}}{%
    \tcbset{width=\mytitlelen}}%
  },
}
\makeatother

\NewTColorBox{receivedmsg}{s o m}{
  commonoptions,
  halign=flush left,
  coltitle=red, colbacktitle=yellow!30,
  colback=yellow!30,
  title={#3},
  IfBooleanTF={#1}{%
    mywidth={#3},
  }{%
    hbox, varwidth upper=\mymaxlen,
  },
  IfValueT={#2}{#2}
}

\NewTColorBox{sentmsg}{s o m}{
  commonoptions,
  halign title=flush right,
  halign=flush right,
  flush right,
  coltitle=blue, colbacktitle=green!30,
  colback=green!30,
  title={#3},
  IfBooleanTF={#1}{%
    mywidth={#3},
  }{%
    hbox, varwidth upper=\mymaxlen,
  },
  IfValueT={#2}{#2},
}

% In your document:
\begin{receivedmsg}{Tostin:}
  You know, I spent a lot of years
    disliking women. But I don't dislike
    you.
\end{receivedmsg}
\begin{sentmsg}*[show bounding box]{Edna:}
  Oh?
\end{sentmsg}
\begin{receivedmsg}{Tostin:}
  You're not a woman. You're more than a
    woman. You're a \emph{mechanic}.
\end{receivedmsg}
```

Tostin:
You know, I spent a lot of years disliking women. But I don't dislike you.

Edna:
Oh?

Tostin:
You're not a woman. You're more than a woman. You're a *mechanic*.

Eventually, I used
`\ifdimcomp{`⟨*dimension expression*⟩`}{`⟨*relation*⟩`}`
`{`⟨*dimension expression*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
from the `etoolbox` package [3] (`tcolorbox` loads it) to compare the width of the title to the maximum width, and, if it is greater, to limit the box width to `\mymaxlen`. I did it using the `tcolorbox` option `width=`⟨*length*⟩, which sets the total width of the box to ⟨*length*⟩.

Please note that `\kvtcb@lefttitle` and `\kvtcb@righttitle` contain the symbol `@`. Thus, the code that uses those names must be included between `\makeatletter` and `\makeatother`. This way, LaTeX will recognize `@` as a letter when looking for the names of commands. Remember to put

`\makeatother` as soon as you no longer need this behavior. Otherwise, you can cause errors, quack!

Since I have to use `mywidth` only when the title is wider than the text, I took advantage of the possibility of testing the starred parameter with
`\IfBooleanTF{`⟨*argument*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`
and use `mywidth` when the starred ⟨*argument*⟩ is present (⟨*true*⟩), or `hbox` and `varwidth upper` otherwise (⟨*false*⟩).

At this point, I've set the dimension, but I also have to indicate the position. The received messages are already located to the left, the default. The sent messages should be on the right, I achieved this with the option `flush right`. Please note that this aligns *the box*, whereas `halign=flush left` aligns *the content* of the box.

The starred parameter seen above is the first `#1` of the customized `tcolorbox` environments in Boxes 5 and 6. The title text is the third `#3`, that I set as mandatory. The second `#2` is optional and I reserved it for any other setting you like to add.

With the command:
`\IfValueT{`⟨*argument*⟩`}{`⟨*true*⟩`}`
you can test if the optional parameter has a value and, if it does, do what is indicated in ⟨*true*⟩.

I added an optional parameter to the environment `sentmsg` of Box 5: `show bounding box`.

It displays the bounding box border around your `tcolorbox`. In the example, you can see that the bounding box has the width of the line, whereas the text box is narrower. In a testing phase, showing the bounding box could be useful. Of course, it should not appear in the final document.

2.4 The finishing touch

The last thing to do is to add a "pointer", to make the box appear as a callout, and the writer's avatar.

You can easily do both things using
`overlay=`⟨*graphical code*⟩.
This allows adding some ⟨*graphical code*⟩ to the box drawing process. See Section 4.12 *Overlays* of the package manual [6] for details.

I added two Ti*k*Z commands, a `\pic` with a Ti*k*Zlings for the avatar and a filled `\path` for the pointer.

For the sake of brevity, I will not explain these macros here; for information, you can refer to the documentation of Ti*k*Z and Ti*k*Zlings packages [5, 7].

What I would highlight is the possibility of using the `frame` coordinates. The `frame` is treated as a Ti*k*Z node. Therefore, you can use `frame.north`, `frame.north west`, and so on, to draw your graphical code.

Let me mention here the option `enlarge left by=`⟨*length*⟩, which enlarges the bounding box distance to the left side of the box by ⟨*length*⟩. Other options for enlarging the right, top, and bottom sides of the box exist as well. See Section 4.15 *Bounding Box* of the manual [6].

In the customized style `myenlarge` of Box 6, the ⟨*length*⟩ is computed as the difference between `\linewidth` and the sum of the widths of the box `\kvtcb@width` and the avatar `\iconwidth`.

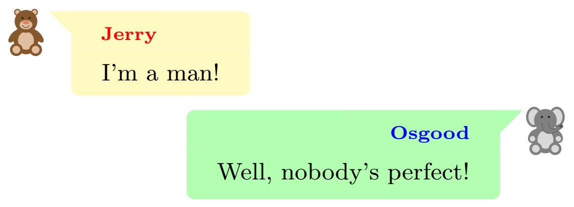
In this case, using `flush right` is not enough, because the avatar is not considered part of the box and, with `flush right`, it would be pushed out of the line border.

Box 6 – Final result

```
\documentclass[twocolumn]{article}
\usepackage[many]{tcolorbox}
\usetikzlibrary{tikzlings}
\usepackage{varwidth}

\newlength{\mytitlelen}
\newlength{\mymaxlen}
\setlength{\mymaxlen}{.9\linewidth}
\newlength{\iconwidth}
\setlength{\iconwidth}{9mm}

\makeatletter
\tcbset{
  commonoptions/.style={
    enhanced,
    fonttitle=\bfseries\scriptsize,
    titlerule=0pt, boxrule=0pt, boxsep=0pt,
    toptitle=6pt, bottomtitle=2pt,
    opacityframe=0,
  },
  myenlarge/.style={
    enlarge #1 by=\linewidth -
    (\kvtcb@width + \iconwidth),
  },
  mywidth/.code={%
  \settowidth{\mytitlelen}{%
    \bfseries\scriptsize #1}%
  \addtolength{\mytitlelen}{%
    \kvtcb@lefttitle}%
  \addtolength{\mytitlelen}{%
    \kvtcb@righttitle}%
  \ifdimcomp{\mytitlelen}{>}{\mymaxlen}{%
    \tcbset{width=\mymaxlen}}{%
    \tcbset{width=\mytitlelen}}%
  },
}
\makeatother
```

```
\NewTColorBox{receivedmsg}{s o m}{
  commonoptions,
  left skip=\iconwidth,
  overlay={
    \pic[scale=.3] at
    ([shift={(-6mm,-6mm)}]frame.north west)
    {bear};
    \path[fill=yellow!30] (frame.north) --
    ([xshift=-3mm]frame.north west) --
    ([yshift=-3mm]frame.north west) --
    cycle;
  },
  coltitle=red, colbacktitle=yellow!30,
    colback=yellow!30,
  title={#3},
  IfBooleanTF={#1}{%
    mywidth={#3},
  }{%
    hbox, varwidth upper=.8\linewidth,
  },
  halign title=flush left,
  halign=flush left,
  IfValueT={#2}{#2}
}

\NewTColorBox{sentmsg}{s o m}{
  commonoptions,
  right skip=\iconwidth,
  overlay={
    \pic[scale=.3] at
    ([shift={(6mm,-6mm)}]frame.north east)
    {elephant};
    \path[fill=green!30] (frame.north) --
    ([xshift=3mm]frame.north east) --
    ([yshift=-3mm]frame.north east) --
    cycle;
  },
  coltitle=blue, colbacktitle=green!30,
    colback=green!30,
  title={#3},
  IfBooleanTF={#1}{%
    mywidth={#3},
  }{%
    hbox, varwidth upper=.8\linewidth,
  },
  myenlarge=left,
  halign title=flush right,
  halign=flush right,
  IfValueT={#2}{#2},
}
```

```
\begin{document}
  \begin{receivedmsg}{Jerry}
    I'm a man!
  \end{receivedmsg}
  \begin{sentmsg}{Osgood}
    Well, nobody's perfect!
  \end{sentmsg}
\end{document}
```



3 Conclusions

I hope you had fun with tcolorboxes and remember, if you have a bug in your code:

Chat with your duck!

References

[1] D. Arseneau. The varwidth package, version 0.92. ctan.org/pkg/varwidth

[2] LaTeX Project Team. LaTeX for authors — current version. ctan.org/pkg/usrguide

[3] P. Lehman, J. Wright. The etoolbox Package: An ε-TeX Toolbox for Class and Package Authors, version v2.5k (2020-10-05). ctan.org/pkg/etoolbox

[4] C. Maggi. The DuckBoat—Beginners' Pond: No more table nightmares with tabularray! *TUGboat*, 44(1):64–70, 2023. tug.org/TUGboat/tb44-1/tb136duck-tabularray.pdf

[5] samcarter. The Ti*k*Zlings package — drawing animals and beings in Ti*k*Z, version 1.0 (2022-06-17). ctan.org/pkg/tikzlings

[6] T.F. Sturm. The tcolorbox package, version 6.2.0 (2024-01-10). ctan.org/pkg/tcolorbox

[7] T. Tantau, et al. The Ti*k*Z and PGF Packages. ctan.org/pkg/pgf

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

Enhancing LaTeX to automatically produce tagged and accessible PDF*

Frank Mittelbach, Ulrike Fischer

This paper was initially presented at the 5th International Workshop on "Digitization and E-Inclusion in Mathematics and Science 2024" (DEIMS 2024) 15–17 February 2024, at Nihon University, Tokyo, Japan. This version contains some minor updates.

The complete program and material on all presentations can be found at the workshop website [1]. A video of the talk and the demonstration is available at `https://youtu.be/7FnZv5FhmRg&?t=9869`.

Abstract

At the TUG 2020 online conference the LaTeX Project Team announced the start of a multi-year project to enhance LaTeX so that it will fully and naturally support the creation of structured document formats, in particular the "tagged PDF" format as required by accessibility standards such as PDF/UA.

In this talk we present the current achievements of this project[1] and the issues we encountered along the way. We also outline open areas of research and the future steps that we shall take to automatically produce well-tagged PDF that supports accessible standards (in particular, the recently finalized PDF/UA-2) as well as general reuse and further conversions. This will be achieved by embedding in the PDF a comprehensive description of the document structure.

Contents

1 General overview

For over 30 years now, the LaTeX system has been used, widely and successfully, for document production in the STEM world and also in other places where high-quality output is required; but until recently its focus was solely on page-oriented output for print (on paper) or as paged output using the PDF format. Therefore, the structural information about the document that was present in the LaTeX source did not get incorporated into the PDF output. Rather, this information was discarded as soon as possible during the processing; this was necessary so as to conserve the limited computer resources (memory and storage) that were typically available at that time (when the core of the LaTeX processing model was first designed).

As long as the intention is only to print a document on a physical medium, then this is all that is required. However, for quite a while now other uses of documents have been increasing in importance so that nowadays many documents are never printed, or printed only as a secondary consideration.

Coming into the 21st century, for many reasons great interest has arisen in the production of PDF documents that are "accessible", in the sense that they contain information to assist screen reading software, etc., and, more formally, that they adhere to the PDF/UA (Universal Accessibility) standard [3, 6], which is explained further in [2].

At present, all methods for producing such "accessible PDFs", including the use of LaTeX, require extensive manual labor[2] during either the preparation of the source or the post-processing of the PDF (maybe even at both stages); and these labors often have to be repeated after making even minimal changes to the (LaTeX or other) source.

1.1 The goals of the multi-year "LaTeX Tagged PDF" project

The main goal of the project is to enhance LaTeX so that it can *automatically* produce tagged PDF without the need to add additional data or commands to the LaTeX source, or to do any of the post-processing work necessary in other workflows.

* This article is a fully tagged and accessible PDF produced by the project software it describes.

[1] This project is carried out by a small number of developers. Besides the authors, the following individuals from the LaTeX Project Team are actively involved: Chris Rowley, David Carlisle, Joseph Wright, Marcel Krüger, and Phelype Oleinik.

We also wish to acknowledge the contributions from various members of the TeX community and beyond; these have been made through comments and suggestions, and more recently through the testing of the new functionality made available in the form of prototype implementations. Without such feedback it would be difficult to finish this project with satisfactory results.

[2] If not using the already existing code extensions to LaTeX provided by the project.

If it remains necessary to alter substantially, or to extend, each individual document in order to provide tagged PDF that conforms to some accessibility standard, then we shall see very few document authors willing to go through the pain of making such additions (unless they are forced to). It is therefore of utmost importance that the generation of tagged PDF be done essentially behind the scenes, with the only cost to the authors being a somewhat longer compilation time.

Another important aim is to make already existing documents accessible by simply recompiling them without the need to alter the source in any substantial way.[3]

The project will support the PDF 2.0 standard [4] (with the very widely supported PDF 1.7 as a fallback solution), because the PDF 2.0 standard offers a more comprehensive tag set, and it supports associated files and many other important features; its use is also a requirement of the new PDF/UA-2 standard [6].

Unfortunately, even though PDF 2.0 has already existed for six years, it has yet to be adopted for industry solutions; e.g., most viewers and other applications are still incapable of making correct use of the new PDF 2.0 features. This is largely a chicken-and-egg problem: because nobody produced 2.0 files, no application was specifically extended to enable processing such files; and due to the fact that no viewer could handle such files, the developers of PDF writers saw no need to invest in the technology to produce PDF 2.0 files.

As a result, LaTeX is one of the first authoring applications that can produce PDF 2.0 files automatically and in large quantities. In particular, LaTeX is capable of producing documents compliant with PDF/UA-2, the new standard for Universal Accessibility [6] that was finalized in 2023 and will be officially released in early 2024.[4] No doubt other suppliers will follow our lead when there is sufficient demand for the production and processing of PDF 2.0 and PDF/UA-2 conformant files.

The document entitled "LaTeX Tagged PDF Feasibility Evaluation" [10], available from the LaTeX Project website [8], explains in detail both the project goals and the tasks that need to be undertaken, concluding with the project plan that is currently being executed.

For the time being the project will focus primarily on PDF output (generated either directly by the TeX engine or through a DVI-based workflow). However, as a bonus outcome of the design approach, the implemented solution will make it easy to add other such output formats to the workflow by simply replacing the output (backend) module. Instead of PDF output, HTML5 or some other format can thus be written. As of now, such alternative backends are not part of the project coverage, but once LaTeX is able, using well-defined interfaces, to pass structure information to a backend, we expect that support for other structured output formats will follow. Such work may be undertaken by us or by other teams, possibly in parallel to later phases of the project.

1.2 Current status and achievements

As mentioned earlier, LaTeX was originally designed, as was essential 40 years ago, to be very economical with computer resources; the implementation therefore worked very hard to discard information as soon as it was no longer needed for the compilation of a document. For print output, which was all that was produced back then, these discards included most of the structural information since this was no longer useful once the visual representation had been determined. An important part of the early work on this project was therefore to alter LaTeX's inner workings by adding code that preserves this structural information from the source and adds it to the PDF.

Another part of this early "background" work was to standardize (and often to provide, for the first time) code interfaces into which extension packages can safely hook. The use of these interfaces, rather than directly overwriting internal LaTeX functions (as was commonly done in the past), avoids the problem that such packages would often break when used in certain combinations, or break when LaTeX internals changed. Moreover, it means that these packages can automatically benefit from the existence of extended workflows (such as those which produce tagged PDF).

Most of these interfaces are now in place in the LaTeX kernel. What remains (as a huge task) is to upgrade many of the core extension packages so that they make use of the new functionalities; this will enable the retirement of some of the existing code that directly overwrites LaTeX internals, or that makes assumptions (about those internals) that will become invalid in the future.

[3] Of course, required data that is not part of the document source (such as alternative text for figures or additional metadata) will need to be manually added, so as to ensure that the document is compliant with PDF/UA. But even if this work is not undertaken, the fact that the document gets automatically tagged will mean that it can be easily navigated and consumed in ways that were impossible before.

[4] At the moment we can only claim that the project software is capable of producing documents that comply with the latest draft of the imminent PDF/UA-2 standard.

The next large phase of the project was to provide automatic tagging for a subset of LaTeX documents. This task is largely finished and therefore most documents that are restricted to using only the commands and environments described in Leslie Lamport's "LaTeX Manual" [7] can be automatically tagged by adding a single configuration line at the top of the document. We say "largely finished" because a few such elements, or element combinations, are not yet covered at the time of writing.

On the other hand, a number of extension packages that go beyond Lamport are already supported, most importantly much of `amsmath` (providing extended math capabilities) and `hyperref` (enhancing LaTeX with interactive hyperlinking features). Also already supported are some of the major bibliography packages, such as `natbib` and `biblatex`.

The project is thus by now capable of producing PDF 2.0 documents that conform to the new PDF/UA-2 standard. In fact, after correcting a small number of issues (not directly related to tagging) in the class file for this conference we have been able to deliver this article as a fully tagged PDF 2.0 document.

1.3 Ongoing and future project tasks

At present, tagging support for the core document elements in a LaTeX document is still at the prototype level, which means that it works for the standard LaTeX classes and for the document elements provided by the LaTeX kernel, but it may or may not work with extension packages or classes that alter the implementation of these document elements, or that provide completely new elements.

To make further progress, some of the interfaces for tagging will first need to be finalized. Then all major extension packages, as well as all important third-party document classes, will need analyzing and possibly updating.[5] The tasks here are to identify all of the legacy low-level code for which the kernel now provides tagging-aware replacements, and then, in cooperation with their maintainers, to make the necessary updates to all these packages and classes.

In addition, any packages and classes that provide new document elements will need to specify how these elements are supposed to be tagged. Some interfaces already exist to help with this process, but it is likely that most of these will require further refinement when tested in the field.

[5] The number of widely used packages, e.g., those described in *The LaTeX Companion, third edition* [9], amounts to roughly 500, so this evaluation and code adjustment forms a substantial part of the remaining project work, and most likely will require additional volunteer support.

The remaining phases of the project, as outlined in the "Feasibility Evaluation Study" [10], cover further support for other PDF standards, and an improved interface to comprehensive metadata. There are also a number of research problems that need to be solved in order for authors to easily generate high-quality tagged PDF documents from their LaTeX sources. These are outlined below.

2 Specific aspects of the project work

We now take a look at a few specific aspects of the project work that are related to challenging problems and pose interesting research questions. These topics are: the development of a more granular tag set; the handling of formulas; the need for an extended table specification syntax; and the handling of language and script related requirements.

2.1 The existing tag set support in PDF

When PDF (already in version 1.3) first introduced a structure tree into the format, to support the inclusion of the document's logical structure, it used only a fairly minimal set of structure tags that were largely modeled after the basic HTML tag set.

For example, for mathematical formulas there was only the `<Formula>` tag itself, with no possibility to add further structure within the formula. For accessibility, all that was available was an "alt" attribute in which one could add a textual description of the formula's content. In a similar manner, other areas of document structures were (over)simplified in the tag set: e.g., for all types of floating elements there is only the tag `<Aside>` that they must share with margin notes (and even that tag is available only in PDF 2.0). For code elements, whether they are small snippets or long, commented listings, there is only a single `<Code>` tag, and there is no option to accurately describe the handling of spaces and new lines within code listings. There is a tag (again only in PDF 2.0) to denote footnotes; but if the document contains several types of structured (and possibly nested) notes, then there is no way to adequately describe this without losing possibly crucial information.

As is also the case with HTML, the relationships between these tags define a fairly simple document model that is not sufficiently rich, so that it cannot express (or not correctly express) many real-life documents; this is often due to the fact that certain elements appear in such documents with nesting relationships that are not permitted by the inclusion rules defined in ISO 32005 [5].

All this means that, when preparing a PDF to be PDF/UA-2 or PDF/UA-1 compliant, compromises have to be made and some of the structural information may thus get lost.

As part of the project we are therefore developing an extended tag set (currently called the "LaTeX namespace") that describes the logical structure of (complex) documents in more granular detail; this will help PDF processors (such as viewers) that understand this namespace to make better use of a document's structure. Ideas from this development may also prove useful in conjunction with future HTML5 developments.

2.2 The LaTeX namespace

LaTeX is an open system that allows for structural extensions (and even changes to structures) in every direction. It is therefore not possible to define a fixed (definitive) document model that is both valid and comprehensive for each and every conceivable LaTeX document.

However, it is possible to define a document model which captures the majority of LaTeX documents that are out there in the real world. If this is combined with methods to extend (and possibly alter) the document model whenever necessary for special structural extensions or changes, we are confident that a comprehensive solution can eventually be provided.

As part of the project we are therefore developing a "standard namespace" that fully describes the LaTeX document model (in the sense outlined above). This tag set will thus be noticeably more detailed and comprehensive than those offered by PDF 2.0 and HTML5. We are working with the PDF Association [11] and various application producers to ensure that this namespace will, when complete, become a recognized resource (preferably acknowledged in future revisions of the PDF standard); it may also be more generally useful as an XML schema. This will, for example, allow PDF and other applications to directly use the extended tag set it provides; and this will enable such applications to make better use of the information contained in the document, whether for accessibility support or for other purposes.

For applications that do not (yet) understand this new namespace, we provide role-mapping back into PDF 2.0 (or PDF 1.7) as necessary; but of course, in that case the more granular information provided by the tags in the new namespace will get at least partially lost.

Additionally, we will be providing interfaces that allow package or class developers who extend the standard LaTeX structures to specify how their new commands or environments map into the LaTeX namespace (and from there, if necessary, are role-mapped back to the PDF tag set).

2.3 Formulas in STEM documents

LaTeX is well-known and appreciated for its ability to describe and format mathematical or other formulas with a high degree of flexibility and unsurpassed quality. This is one of the reasons why we see a huge proportion of the documents in STEM disciplines such as mathematics, physics, and computer science being produced using LaTeX.

As described by Neil Soiffer in his keynote for the DEIMS 2021 conference [12], there are basically three methods for making such formulas accessible in a tagged PDF. One option is to use static text that can be attached as "alternative" natural language text to the formula structure, which is then read by an "assistive technology" (AT) application. While rather easy to implement, this method has various drawbacks: it does not allow for braille generation or for exploration of the equation; and the text often must be hand-crafted to avoid problems with reading software whose heuristic usually ignores certain symbols such as punctuation or braces.

A second option is to add marked-content operators to the PDF stream and then build a MathML structure tree that references this marked content. This method leads to a large structure tree with many objects, since this tree will be very fine-grained.

There are a number of problems with this second approach. It is difficult for LaTeX (without the help of a real programming language) to generate correct and useful MathML while building the TeX math list. Furthermore, when the still widely used pdfTeX engine is producing the PDF, there is a high chance that the combined processes (of simultaneously adding the necessary tagging-related material to the content stream while formatting the formula) will alter the spacing of the formula and thus render the visual representation invalid. There may be technical solutions to circumvent these issues and this is an area of active research. However, it is likely that this option can only be implemented successfully if the LuaTeX engine is used, because then a suitable programming language (i.e., Lua) is available and, furthermore (because of extended functionality in LuaTeX), it becomes possible to delay adding the necessary extra material to the content stream until after LaTeX has completed the formatting of the formula with the correct spacing.

The last option is to make use of so-called "associated files" (AF) that were introduced in PDF 2.0: these are files directly embedded into the PDF that can be attached to a structure element.[6] Each such "embedded AF" can contain, for example, a MathML representation for a formula, or its LaTeX source or some additional commentary text; more than one of these can be attached to each structure. The AF approach is simpler and easier to implement, and it also allows the use of MathML representations that do not closely follow the visual output; but it has the drawback that the MathML in the AF file is associated only to the formula as a whole and it is therefore not possible to synchronize parts of the MathML representation with the corresponding parts of the formula in the typeset document, as is necessary to support navigation of formulas and highlighting them. This method may require that the AT software overlays the printed output with its own rendering of the MathML (which may differ substantially from the original rendering).

The two last options, MathML in the structure tree or in associated files, both suffer from a lack of support in current PDF viewers and AT software: Neil Soiffer's optimistic statement in 2021 that "*Adobe's API will likely incorporate this ability in the future*" has not yet come true.

Because of this, LaTeX currently follows a threefold strategy in the prototype for math tagging: it incorporates the LaTeX source as alternate text for the formula, under the assumption that the LaTeX syntax is understandable to most readers of mathematics; and it also embeds the LaTeX source as an associated file. Additionally, an external file can be constructed in which, for all (or a selection of) the formulas, a MathML representation is provided that can be embedded in the PDF as associated files. Such an external file can be created, for example, with the help of `tex4ht` or with the LuaTeX engine. At this point in time there is no fully automatic workflow implemented for this, but with only a few adjustments it was already possible to add MathML associated files to all the formulas in the `amsmath` user documentation.

The form of the final solution for formulas, and whether or not it is necessary to offer customizable alternatives — to cater for different reader deficiencies or different user preferences — are questions that need active research to understand how to best serve consumers given the currently limited functionality of AT tools with respect to "associated files", etc.

There is another aspect of common LaTeX usage that affects all three of these method, and is also found in many other areas beyond formulas and STEM: the inclination of authors to invent new symbols, notation systems, and command names. This is nowadays exacerbated by the widespread failures to take accessibility into account. Such ad hoc extensions make it difficult to fully automate any tagging process. Overcoming this will need both technical support for such extensions and also, perhaps more importantly, encouragement of authors to keep accessibility in mind when writing documents.

The approach that will most likely be adopted to deal with this issue is as follows: by default, assume that new commands are simply abbreviations and that, by recursively replacing each of these with its definition, we eventually get to something that can be automatically tagged by using standard methods. For cases where this does not work, there will be interfaces with which the author of the document (or the package developer, if the command is defined there) can specify how the command should be interpreted when providing tagged output (e.g., MathML).

2.4 Extensions to LaTeX's table handling

In most cases, the LaTeX source will contain all the necessary information about the logical structure of a document, so that it is possible to automatically transform the source into richly tagged PDF output. There is one noticeable exception: LaTeX's handling of tabular data. This arises since standard LaTeX, and most extension packages, do not describe table data through structural information; rather, they do this in a purely visual fashion, describing only the content that should go into each cell. Thus no information is supplied concerning important relationships between cells, such as which are the header or sub-header cells, or to which cells some header cell applies.

Thus, while it is fairly trivial to tag tables as simply consisting of table rows and table data cells, determining the header cells can be done only by the use of heuristics (e.g., cell formatting changes done through `\multicolumn` are likely to represent header cells, or certain rules in a table may indicate header rows). However, any such heuristic will have a noticeable number of counterexamples.

It is therefore an important task to develop good heuristics that correctly cover a large proportion of the tables in legacy documents; and in addition to develop a syntax extension for LaTeX that allows authors to specify such logical structure explicitly in case the heuristics fail or they wish to specify explicitly the logical structure of the table. This syntax extension has to be done in a lightweight way, i.e., without putting an unnecessary burden onto

[6] Note that "associated files" do not exist as separate physical entities at the operating system level; thus they are not in fact "files" in the normal sense of the word.

the authors. Furthermore, it should be upwardly compatible with the existing syntax so that it is easily possible to enhance documents with only small alterations to the original source.

It is also important to develop methods that enable authors to easily check LaTeX's interpretation of the logical structure of a table without the need to examine the final PDF, so that they can overwrite the heuristics when necessary. This is an area of active research.

2.5 Support for different scripts and languages

Historically, the TeX engine and LaTeX were developed for ASCII-based, English documents and then (with TeX 3.0) extended to support other languages and scripts — at first, because of the restrictions to 8-bit codepages, mostly for languages using Latin scripts, but later also to non-Latin scripts, such as Greek or Cyrillic, as well as more diverse scripts. Initially, the solutions for all such scripts required complex font setups (as done, e.g., by the `CJK` package), special processors to handle transliterations, and engine extensions to handle, for example, right-to-left scripts or special input encodings.

The advent of Unicode and the Unicode-aware engines (XeTeX, LuaTeX and upTeX) led to the existence of simpler, and much more powerful, setups; therefore, most scripts are now well supported in LaTeX — perhaps with the exception of scripts that change the writing direction, since this isn't part of the original LaTeX design and thus often requires overwriting many standard commands.

The project currently concentrates on documents that use Latin scripts or scripts with similar characteristics. The correct tagging conventions to use with other types of script are not yet known by us: e.g., how to deal with direction changes or ruby characters. When using scripts (such as Latin) that typically use "whitespace" to delimit "words", tagged PDF has a requirement that even within the typeset content stream these words must remain delimited by an explicit "whitespace character" [4, §14.8.2.6.2]. This conflicts with the normal practice of TeX typesetting engines since they do not naturally add such delimiter characters; however, both pdfTeX and LuaTeX have been modified to provide workarounds for this.[7] We also do not yet know to what extent the many external packages supporting diverse scripts and languages will need to be adapted for the support of tagging. To research these topics, help from users and developers with in-depth knowledge of such scripts will be needed.

[7] Engines, such as XeTeX, that do not offer this workaround can therefore not be used to produce PDF/UA documents involving scripts that separate words with spaces.

3 Some TeXnical details

In this final section we take a brief look at two technical aspects of the project work. Both will be covered in more depth during the demonstration session at the conference.

The first subsection explains how to set up a document (such as this one) so that it will automatically produce tagged PDF. This should, we hope, enable you to immediately experiment with the addition of such tags to your own works. If you want to provide feedback on any issues that you encounter, or to provide suggestions for improvements, we suggest adding them to the project repository `https://github.com/latex3/tagging-project`, using either the `issues` or the `discussions` page, as appropriate.

This is followed by some background information on the experiments we are currently conducting to automatically include in the PDF MathML representations of all the formulas in a document. We expect this work to become available for public testing during 2024/Q2.

3.1 How to enable tagging

Until recently there was no dedicated location in LaTeX documents to declare settings that affect the document as a whole. Settings had to be placed somewhere in the preamble or as class options, or sometimes even as package options. For some such settings this was problematic, e.g., setting the PDF version is only possible if the PDF output file has not yet been opened, which can be caused by loading one or another package. For the "LaTeX Tagged PDF project" [10, p. 17] further metadata about the whole document (and its processing) needs to be specified, and again all this data should be placed in a single well-defined place.

For this reason we introduced (in June 2022) the command `\DocumentMetadata` so as to unify all such settings in one place. This command takes one argument that should contain a key/value list specifying all the document metadata for the current document.[8] This should be placed at the very beginning of the document, i.e., *before* `\documentclass`; it will produce an error if found later.

The `\DocumentMetadata` command also loads the LaTeX PDF management bundle, which provides various PDF-related commands that are needed to

[8] At this point in time only a few keys are accepted, e.g., to set the PDF version, the language, a PDF standard and to load a color profile.

```
<div>
<h2>\mml 65</h2>
<p>\begin{math}\sqrt [\beta ]{k}\end{math}</p>
<p>656E4D3BB4F29D20A1B2CBCB35C35E7E</p>
<math xmlns="http://www.w3.org/1998/Math/MathML" display="inline">
<mroot>
<mi>k</mi>
<mi>&#x03b2;</mi>
</mroot>
</math>
</div>
```

Figure 1: Sample entry with MathML data for associated file (AF)

create a tagged PDF. It also accepts the `testphase` key, which is of a temporary nature since it is needed only while new functionality is being introduced for testing. This key is used to load specific tagging support: this article, for example, uses the following:

```
\DocumentMetadata{
   testphase={phase-III,table},
   pdfversion=2.0,
   pdfstandard=a-4,
}
```

which loads the tagging support from `phase-III` (basic document elements) and `table` (newly developed prototype code for tagging of `tabular`-like environments not yet integrated in any test phase). In addition, the PDF version is set to 2.0 and it specifies that the PDF should be compliant with the PDF/A-4 standard. This is all that was necessary to produce the tagged version of this article.[9]

Eventually, the testphase code will move (once all components are considered stable) into the LaTeX kernel itself and the `testphase` key will vanish. Tagging will continue to require a `\DocumentMetadata` declaration, but will then use a simple `tagged=true` key (name to be decided).

3.2 Inclusion of external MathML

As outlined in section 2.3 on page 56 we are currently experimenting with a scheme in which externally provided MathML is embedded in the PDF as AFs. The MathML for the formulas is provided in an external file containing one or more `\mml` commands with the format shown in figure 1 (i.e., surrounded by HTML tags so that it can be proofread in a browser).

The first argument to `\mml` is a label (e.g., a number) to that uniquely identifies this MathML snippet; the second argument contains the LaTeX source for the MathML. The third argument is the MD5 hash of the LaTeX source. Its use ensures that the PDF file will contain only one AF for any formula, even if a formula is repeated several times: for example, if the LaTeX source document repeatedly uses `$\beta$`, then each repetition of exactly this formula will reference the same embedded AF, which means that the PDF file does not become unnecessarily large.

The final argument contains the corresponding MathML. In our current experiments the MathML is generated from the LaTeX source by processing it with `tex4ht`, with some further processing to add the MD5 hash values and with some manual corrections to improve the resulting MathML. One advantage of using an external file at this stage is to allow the MathML to be validated before being embedded as an AF in the PDF. The MathML could potentially be generated by other TeX to MathML conversion programs such as `latexml` or `luamml`, which would allow experimentation with different pipelines to construct associated files containing MathML.

This file is then input at the beginning of the document and each MathML (with a unique hash value) is embedded in the PDF as the content stream of an AF. The LaTeX code to produce tagged PDF then checks, for each math formula in the document, whether an associated file containing MathML for this formula has already been added to the PDF, and, if so, a reference to this MathML associated file is added to the `<Formula>` structure element being constructed. Therefore, if the same math expression occurs more than once (as a complete formula) then each occurrence will reference this same MathML AF.

Currently, the generation of the file of MathML fragments requires some manual editing and explicit execution of conversion programs. The next step will be to create scripts that will: run directly in a LuaTeX compilation; fully automate the generation of the MathML fragments from the LaTeX source; and validate this output.

[9] This paper does not contain any tabular material, thus `table` is actually unnecessary for tagging this article. The setting was added to show how the interface can be used when new functionality is made available.

References

[1] DEIMS 2024. Website of the *5th International Workshop on Digitization and E-Inclusion in Mathematics and Science 2024*, Nihon University, Tokyo, Japan, February 2024. `workshop.sciaccess.net/deims2024/`.

[2] Olaf Drümmer and Bettina Chang. *PDF/UA in a Nutshell — Accessible documents with PDF.* PDF Association, August 2013. `pdfa.org/resource/pdfua-in-a-nutshell/`.

[3] *ISO 14289-1:2014; Document management applications — Electronic document file format enhancement for accessibility — 1: Use of ISO 32000-1 (PDF/UA-1)*, 2nd edition, 2014. `www.iso.org/standard/64599.html`.

[4] ISO. *ISO 32000-2:2020(en); Document management — Portable document format — Part 2: PDF 2.0*, 2nd edition, 2020. `iso.org/en/contents/data/standard/07/58/75839.html`.

[5] *ISO/TS 32005:2023; Document management — Portable Document Format — PDF 1.7 and 2.0 structure namespace inclusion in ISO 32000-2*, 1st edition, 2023. `iso.org/en/contents/data/standard/04/58/45878.html`.

[6] *ISO/FDIS 14289-2; Document management applications — Electronic document file format enhancement for accessibility — Part 2: Use of ISO 32000-2 (PDF/UA-2)*, 1st edition, 2024. `www.iso.org/standard/82278.html`.

[7] Leslie Lamport. *LaTeX: A Document Preparation System: User's Guide and Reference Manual.* Addison Wesley, 2nd edition, 1994.

[8] LaTeX Project Team. Website of the LaTeX Project. `latex-project.org/`.

[9] Frank Mittelbach and Ulrike Fischer. *The LaTeX Companion.* Addison-Wesley, Boston, MA, USA, third edition, 2023.

[10] Frank Mittelbach, Ulrike Fischer, and Chris Rowley. *LaTeX Tagged PDF Feasibility Evaluation.* LaTeX Project, September 2020. `latex-project.org/publications/indexbyyear/2020/`.

[11] PDF Association (PDFA). Website of the PDF association. `pdfa.org/`.

[12] Neil Soiffer. Accessible PDF: 2 $\not>$ 1. In *The 4th International Workshop on Digitization and E-Inclusion in Mathematics and Science 2021.* The DEIMS2021 Organizing Committee, 2021. `workshop.sciaccess.net/deims2021/DEIMS2021_Proceedings.zip`.

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

◇ Ulrike Fischer
Bonn, Germany
`https://www.latex-project.org`

Preparing Horizon Europe proposals in LaTeX with `heria`

Tristan Miller

Abstract

This article introduces `heria`, a LaTeX class to format funding proposals for the European Commission's Horizon Europe program. It provides a basic summary of the class's use; compares it to existing packages for funding proposals; discusses its motivations, design decisions, and limitations; and reports on its real-world use and plans for future development. Besides providing prospective Horizon Europe applicants with an overview of the class, this article may give prospective developers and users of classes for other proposal types some idea of the work involved and the potential pitfalls.

1 Introduction

Horizon Europe is a seven-year, €95.5 billion initiative of the European Commission (EC) that is intended to fund research and innovation projects in the European Union and its wider network of global partners. The EC earmarks portions of the total budget according to various topics and action types, issues calls for proposals of projects supporting those topics and action types, and then disburses the funds to applicants according to a competitive evaluation process. The types of projects solicited often require a large consortium of partners — potentially dozens — and the calls prescribe a specific, intricate structure for the proposals, which can run to hundreds of pages. While the EC does not require applicants to use any particular content authoring tool, the only templates it distributes are in Rich Text Format (RTF) — hardly the most convenient format for multiple authors to collaborate on producing a lengthy, heavily (cross-)referenced technical text.

This article introduces `heria`, a LaTeX class to format proposals for the Research and Innovation Actions (RIA) and Innovation Actions (IA) of Horizon Europe. Using `heria` and a networked source control system or collaborative online LaTeX editor, it becomes easier for a dozen or more authors to jointly produce an elegant, internally consistent proposal conforming to the EC's requirements. Unlike the default RTF template, the `heria` class manages the numbering of and references to project elements (participants, work packages, etc.) as they are added and removed, and programmatically regenerates and resums the requisite data tables (for staff effort, project costs, etc.). This helps ensure that data changed in one part of the proposal remains consistent with explicit and implicit references to it elsewhere in the proposal. The class also preserves the instructions from the original template, but allows users to toggle their visibility, either individually or *en masse*, so that instructions can be hidden once they are fulfilled, or once the proposal is ready to submit.

Besides providing a very basic summary of how `heria` is meant to be used, this article compares it to existing packages for funding proposals; discusses its motivation, design decisions, and limitations; and reports on its real-world use and plans for future development. This material should help prospective Horizon Europe applicants decide whether it makes sense to use `heria` for their proposal; perhaps equally importantly, it may inspire others to develop and publish their own packages for other types of funding proposals (whether for Horizon Europe or some other funding scheme) and it may give them some idea of the work involved and the potential pitfalls.

2 Previous proposal packages

Perhaps surprisingly, given the TeX ecosystem's popularity among academics and technologists, CTAN boasts only a handful of packages for typesetting research funding proposals. The `nih` package [2], last updated in 2005, provides a LaTeX class to format grant applications to the US National Institutes of Health (NIH). The more recent `grant` package [7], last modified in 2019, also handles LaTeX proposals for NIH, as well as five further American agencies, including the National Science Foundation and the Defense Advanced Research Projects Agency. Neither package provides much in the way of formal documentation, though `nih` at least comes with substantial example documents. The `mynsfc` package [10], last changed in 2020, provides a XeLaTeX class for proposals to the National Natural Science Foundation of China (NSFC). As with the previous two packages, there is little or no technical documentation concerning how to use the class; however, the package does reproduce the NSFC's instructions concerning the content and formatting of the proposal.

The `proposal` [8] and `h2020proposal` [6] packages are seemingly the only ones until now that specifically target the preparation of European grant proposals. The most recent CTAN release of `proposal` dates to 2016, but development has continued on the project's GitHub repository at `github.com/KWARC/LaTeX-proposal`. The package includes LaTeX classes for proposals to the German Research Foundation and the EC's Framework Programme 7 (FP7), along with documentation and examples. The `h2020proposal` package, dating to 2015, contains classes for RIA proposals in the EC's Horizon 2020 program, the predecessor of Horizon Europe. The templates are set up for use with LaTeX but contain guidance on adapting them for use with XeLaTeX. As with `proposal`, `h2020proposal` includes documentation and examples, and like `mynsfc`, the templates helpfully reproduce the funding agency's content- and formatting-related guidelines.

Although these last two packages are fairly elaborate, and (according to their authors) have even been used to prepare real proposals for submission to the EC, they share two significant shortcomings. First, neither package supports the proposal format used by Horizon Europe, the current EC framework program that runs from 2021 until 2027. (FP7 ran from 2007 to 2013, and Horizon 2020 from 2014 to 2020.) Second, both packages are admittedly incomplete: the documentation for `proposal` indicates that the package is "relatively early in its development", and the documentation for `h2020proposal` warns would-be users that it is "still in a beta-testing stage" and should not be distributed.

3 Motivation and design decisions

The principal motivation for producing a new proposal class, rather than extending an existing one, was to support applications to the current Horizon Europe RIA and IA actions. Although `proposal` and `h2020proposal` have many good ideas in terms of their implementation, the historic proposal types they support differ so much in terms of form and content from current ones that it would be very challenging to extend or even adapt these packages. The `heria` class was therefore written entirely from scratch, though it does draw some inspiration from the interface of `h2020proposal`.

The Unix philosophy was another influence on `heria`, or more specifically its maxim, "Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new 'features'." [9] The two aforementioned packages include some extra bells and whistles that are not strictly necessary for preparing a proposal. Besides supporting both European and German proposals, `proposal` includes functionality for preparing grant agreements and final project reports, which are significantly different in structure and formatting. And both packages provide a mechanism for generating Gantt charts for use in the proposal; while the official Horizon Europe application instructions require applicants to indicate the "timing of the different work packages and their components", it does not require this to be in the form of a Gantt chart. In any case, in the present author's experience, how best to style and structure a Gantt chart varies greatly from project to project, and so having heria provide its own implementation would pose a number of challenges. Either the implementation could be a simple one targeting the lowest common denominator, which many users would find limiting, or it could allow for great versatility in the design of the chart, in which case it could end up as little more than a wrapper for the pgfgantt package [12]. Since pgfgantt already exists and is fairly simple to use, heria assumes that proposal authors will simply use that if they want a Gantt chart.

It was also important that heria be easy to use. Like proposal and h2020proposal, the macros it provides are fairly simple and often store information for use later in the proposal, and like h2020proposal, it helpfully exposes the official application instructions to the user. With seven pages of prose, the documentation for heria is comparable in breadth and depth to that of h2020proposal; direct comparison with proposal is not practical owing to the latter's goal of supporting more funding schemes and non-proposal document types.

A final design decision that was of great importance to the developer was that the package should allow users to produce a proposal using only free software [4]. The heria package is therefore released under the terms of the LaTeX Project Public License. The packages it depends on, as well as LaTeX and TeX themselves, are also available under various licenses permitting free use, modification, and (re)distribution. While proposal co-authors may choose to collaboratively edit their heria-based proposal using a proprietary online service such as Overleaf, they could alternatively use an entirely free authoring pipeline with a source control system such as Git.

4 User interface

The intention of this section is not to recapitulate the complete package documentation, but rather to give a very general overview of heria's interface and features. This serves as an introduction to prospective users and as context for some of the observations and discussions found later in this article.

The heria package consists of a class file (heria.cls), a set of LaTeX files containing the application instructions, a skeleton proposal (heria-example.tex), and the package documentation (heria.pdf). Since the official Horizon Europe proposal template requires proposals to follow a fairly strict and detailed structure, the best way of starting a new proposal is to make a copy of the skeleton proposal and then adapt it by replacing its dummy data and supplying any missing information.

As in the official template, lengthy application instructions are interspersed throughout the skeleton proposal. Most of these instructions are emitted via an \heinstructions macro at the appropriate place in the proposal document; the argument to this macro specifies one of the aforementioned files containing the instruction text. Other instructions take the form of recommended page limits printed next to section headings; these limits are specified via an optional argument to the \section, \subsection, and \subsubsection macros. Instructions are printed only when the showinstructions option is passed to \documentclass; instructions can alternatively be omitted on a case-by-case basis by removing or commenting out the corresponding \heinstructions macro (or option to the section heading macro, as the case may be).

The class, which is derived from the standard LaTeX article class, takes care of setting the fonts, margins, etc. as mandated by the official template, and redefines some common commands (\maketitle, \section, etc.) to produce titles and headings in the prescribed format. This includes the so-called "tags" (cryptic identifiers such as "#§CON-MET-CM§#") that the official template places around certain section headings and warns applicants not to move, remove, or change in any way.

For many parts of the proposal, applicants can simply provide free-form text, along with whatever lists, tables, figures, etc. they think necessary. For other parts, the official template requires applicants to provide information in a fixed format, usually corresponding to one or more tables. Often these tables, and/or the rows or columns within them, must be printed in a particular order that is determined by information entered elsewhere in the proposal. For example, the table summarizing the staff effort has one row for each participant in the project, and these rows must be arranged in the same order as in the earlier table listing the participants; it also has one column for each work package in the project, and these columns must be arranged in the same order as the earlier table listing the work packages. Besides this, the staff effort table needs to include a final row that sums the numbers in each column, and a final column that sums the numbers in each row. To obviate the need for users to tediously re-arrange and re-sum such tables every time a participant or work package is added, removed, or re-ordered, heria provides (a) macros such as \participant and \workpackage for defining participants, work packages, and other project data in such a way that heria remembers their original order and that users can explicitly reference them in subsequent macros and environments, and (b) macros such as `\makeparticipantstable` and `\makeworkpackagestable` that automatically generate the data tables in the correct order, and with automatically computed sums, on the basis of the order and content of the previous definitions.

Perhaps the most typographically complex part of the official template is the "summary canvas", a tableau of framed text boxes that is spread across the whole of one or two pages with landscape orientation. The `heria` class provides `summarycanvas` and `summarybox` environments for typesetting these boxes, as well as a `SidewaysFigure` environment that takes care of rotating the page in a way that preserves readability in PDF viewers.

5 Limitations and workarounds

The official RTF template has a number of oddities and limitations, and in adapting it to LaTeX it was necessary to decide, on a case-by-case basis, whether to preserve or work around them. The following points discuss some of these decisions:

Vague instructions. Some of the official instructions for filling in the tables admit of more than one possible interpretation. Perhaps the only such ambiguity with bearing on `heria`'s behaviour concerns how purchase costs in a given category are to be listed in the associated table; it is not clear whether applicants should itemize these costs across separate rows or combine them into a single row. The `heria` class takes the latter interpretation, though future releases might include support for itemized costs.

Tables that aren't tables. Many of the proposal elements that the official template refers to as "tables" are not, typographically speaking, tables, nor even what some dismissively refer to as "tableaux" [3]. For example, the template's Table 3.1b, headed "Work package description", is actually a 2×2 tableau for entering the work package number and title, followed by two separate, framed, full-width paragraph boxes for entering the work package's objectives and description of work. These three elements are to be repeated, all under the same "Table 3.1b" caption, for every work package. In a typical proposal, Table 3.1b will have content running across several pages, and almost none of it will be tabular. All such "tables" in the official template are therefore adapted into `heria` as LaTeX subsections, with custom macros for the user to provide the "table" data and another custom macro to finally output it in the prescribed format, using an appropriate combination of `tabular`-style environments and framed boxes.

No provision for floats. The RTF format has little or no support for floating objects; the official template is therefore written with the expectation that all the required information will be presented in a linear fashion and in the prescribed order. This can pose problems when there is not enough room remaining on a page to typeset a table; the table would normally have to start at the top of the next page, leaving wasted space on the previous page. The `heria` class solves most such problems with measures that allow tables to gracefully break across pages. The one exception is the template's landscape-oriented "summary canvas" that forms the sole content of Section 2.3. The page rotation precludes any possibility of beginning or ending the tableau on the same page as the preceding or following material, respectively. Here the skeleton proposal distributed with `heria` makes an arguably justifiable departure from the official template by putting the summary canvas in a floating figure, and then adding a one-line reference to it under the Section 2.3 heading. While this may not be strictly in line with the official template, it at least averts the danger of having the canvas introduced by a page that, except for the section heading, is nearly or entirely blank.

Other wasted space. Besides the lack of floats, there are other cases in which the official template doesn't make or even allow for efficient use of space. For example, the 2×2 work package tableau mentioned above seems altogether gratuitous, since the information it contains could easily have been combined into a single line. In other cases, bona-fide tables are given needlessly verbose column headings that introduce extra line breaks or steal horizontal space from the other columns. The `heria` version of the template preserves the original's gratuitous structural elements (since the presence of these may be subject to formal checks by the funding agency) but takes some liberty in slightly abbreviating, or at least hyphenating, some words in table headings.

Besides these imposed limitations, the package has a few shortcomings that are down mostly to the rushed initial development. For one, the class does very little specialized error checking on its input. Usually passing an invalid argument to one of its macros, or neglecting to provide data necessary to generate a table, will result in some sort of compilation error, though the diagnostic message emitted may be somewhat obscure. Another issue is that the class generally expects users to supply numeric data (for person-months, costs, etc.) as integers. Though decimal arguments to certain macros may be correctly interpreted, and some provision has been made for the class to use floating-point arithmetic when calculating sums, the code that emits numbers in generated tables cannot be relied upon to produce elegant output. Solving both these issues is on the agenda for future development.

6 Reflections and case study

The process of developing `heria` proved to be rewarding and frustrating in equal measures. On the one hand, it presented the developer with the motive and opportunity to apply and extend his LaTeX programming skills, and bestowed upon him an intimate familiarity with the application requirements for an active proposal submission (described below). On the other hand, having to reproduce and stay within the aesthetic and structural limitations of the official RTF template felt unduly constraining, particularly when those limitations seemed to be the product of questionable or even deleterious design decisions. This agony would perhaps have been felt less acutely had the proposal co-authors decided to forgo the use of LaTeX in favour of a less capable tool.

Even still, LaTeX itself was also at times a source of consternation when developing `heria`. The sort of high-level programming required to *easily* automate the management of proposal data and the generation of data tables is not well supported by LaTeX: many of the basic data structures necessary for these tasks, and the basic algorithms for accessing, sorting, and iterating over them, are either not present in the language, or require obscurely named and relatively under-documented LaTeX3 macros, or are implemented only in third-party packages that must first be discovered and then learned. Of course, these criticisms of LaTeX are hardly new (see, for example, [1, 5, 11]). In hindsight, it may have been a better idea to write the class in LuaTeX, even at the cost of having to learn it (and Lua itself) from scratch.

On the whole, the initial development of `heria` probably took about as much time as it would have taken the coordinator of a large word-processed proposal to manually resolve all the edit conflicts, formatting problems, bibliographical inconsistencies, and outdated cross-references introduced over the entire writing process. Anyone considering developing a LaTeX class for proposals for another funding program should therefore consider whether it makes sense to invest the effort; if the template is unlikely to be used more than once, then it may be better to hold one's nose and use the official version.

`heria` saw its first real-world use case in 2023, for a highly interdisciplinary Horizon Europe RIA proposal co-authored by 19 people across 14 organizations in ten countries. The organizations included universities, an independent research institute, several small businesses, and branches of a multinational company. Many of the co-authors held degrees in computer science, but others came from the social sciences or humanities. Accordingly, they varied greatly in their prior knowledge of LaTeX, from none at all up to several decades' experience.

The proposal document was hosted on Overleaf, which allowed co-authors to edit it online or to check it out via Git for offline editing. According to the document's edit history, there were 21 403 distinct edits made, of which 20 631 (96%) were carried out online in Overleaf and 4% were committed through Git. It should be noted, however, that Overleaf's tracking of changes is considerably more fine-grained than Git's. Someone writing offline might produce several pages' worth of material and then submit it in a single commit to the Git repository, but had the same material been entered directly into Overleaf, the service may have recorded this as hundreds of distinct changes. It should also be borne in mind that the package developer was among the co-authors, and about 10% of the Git commits included updates to the `heria` class itself.

At the time, there was no formal documentation for `heria`; the other co-authors were provided only with a lightly commented skeleton proposal and a 300-word `README` explaining how to compile it, add citations and to-do notes, and toggle the visibility of the instructions. Nonetheless, the writing process proceeded smoothly, with the package developer receiving virtually no questions of a TeXnical nature, even from the LaTeX neophytes. The writing process exposed a few bugs in the package code, which were duly fixed, and also provided the impetus for a few optimizations and aesthetic improvements. The present author was among those who helped perform an internal consistency check of the final draft of the proposal, and found considerably fewer issues than in a past experience with a Horizon proposal written in Microsoft Word. In the end, the `heria`-formatted proposal was submitted on time; it passed all formal checks by the funding agency and so was duly forwarded to the reviewers. It is either great modesty or great shame that prevents the author from revealing the final accept/reject decision here, though for our purposes it suffices to say that the use of `heria` played no direct part in it.

After the proposal was submitted, the developer informally surveyed its other 18 co-authors for their feedback on the writing process insofar as it related to using LaTeX and Overleaf in general and `heria` in particular, and asked them to compare the experience with those for any past proposals collaboratively written with different tools. Eight responses were received and all indicated a positive experience with the heria-based workflow. Four respondents specifically highlighted heria's capacity to enforce consistency in the proposal's structure, formatting, and/or references; three respondents expressed appreciation or enjoyment at being able to leverage their existing LaTeX knowledge; and two thought that the package enhanced the group's ability to edit collaboratively.

Several responses described past experiences with Microsoft Word, Microsoft 365, and Google Docs as being inefficient or even "painful", indicating that "formatting will be a mess with many people collaborating". Nonetheless, they recognized that these tools are more familiar to those outside computer science, and so may have a lower barrier to entry. They also praised Google Docs's commenting facility, which allows co-authors to annotate documents with tasks, to assign them to individual collaborators, and to receive email notifications whenever tasks are created, replied to, or resolved. The heria-based workflow had no comparable commenting facility; co-authors used a mixture of comments in the LaTeX source code, in-document comments typeset with the todonotes package, and Overleaf's own commenting feature. This proved to be problematic, since the source code comments were not always visible to co-authors using Overleaf's visual editor and the Overleaf comments were not visible to co-authors who checked out the project with Git to edit it offline. Since providing a general-purpose issue tracking system is well beyond the scope of heria, anyone considering heria (or any other LaTeX-based workflow) for a large collaborative project should therefore consider how best to coordinate tasks and discussions during the writing process.

7 Availability and future development

The heria package has an official website at logological.org/heria that includes links to its documentation, source code repository, and bug tracker. The package saw its initial release to CTAN on December 4, 2023, and it was added to TeX Live the following day. By the time this article is published, heria may also be available in other TeX distributions and online editors.

From time to time, the EC revises the official template for Horizon Europe RIA and IA proposals, and it is intended that heria, over the course of its development, will track these revisions. Whether this intention is realized depends on the availability and motivation of its maintainers, a group that for the moment consists solely of the present author. Anyone interested in directly contributing to the further development of heria is welcome to get in touch. Failing that, the best possible impetus for continued improvement of the package is the opportunity for the author to participate in further Horizon Europe proposals. (Readers are welcome to take this as a coy solicitation to collaborate on high-quality research projects.)

References

[1] N.H.F. Beebe. 25 Years of TeX and METAFONT: Looking back and looking forward — TUG 2003 keynote address. *TUGboat* 25(1):7–30, 2004. tug.org/TUGboat/tb25-1/beebe-2003keynote.pdf

[2] B. Donald. The nih package, 2005-06-01. ctan.org/pkg/nih

[3] D. Els, S. Fear. The booktabs package, version 1.61803398, 2020. ctan.org/pkg/booktabs

[4] Free Software Foundation. What is free software?, Feb. 2021. www.gnu.org/philosophy/free-sw.en.html

[5] H. Hagen. LuaTeX: Howling to the moon. *TUGboat* 26(2):152–157, 2005. tug.org/TUGboat/tb26-2/hagen.pdf

[6] G. Indiveri. The h2020proposal package, version 1.0, 2015-09-20. ctan.org/pkg/h2020proposal

[7] J. Karr. The grant package, version 0.0.5, 2019-02-26. ctan.org/pkg/grant

[8] M. Kohlhase. The proposal package, version 1.5, 2016-04-15. ctan.org/pkg/proposal

[9] D. McIlroy, E.N. Pinson, B.A. Tague. Unix time-sharing system: Foreword. *The Bell System Technical Journal*, 57(6):1899–1904, July–Aug. 1978. archive.org/details/bstj57-6-1899

[10] F. Qi. The mynsfc package, version 1.30, 2020-08-18. ctan.org/pkg/mynsfc

[11] R. Reich. Does TeX/LaTeX give a headstart with other programming languages? [answer], Jan. 2012. tex.stackexchange.com/a/42749/22603

[12] W. Skala. The pgfgantt package, version 5.0, 2018. ctan.org/pkg/pgfgantt

◇ Tristan Miller
Department of Computer Science
University of Manitoba
Tristan.Miller (at) umanitoba dot ca
https://logological.org
ORCID 0000-0002-0749-1100

Specifying and populating documents in YAML with `lua-placeholders` in LaTeX

Erik Nijenhuis

Abstract

This article examines the implementation of the invoice template in GinVoice [3] and explores how the invoice template can better align with the LaTeX ecosystem by introducing an additional data layer in YAML using `lua-placeholders`. With the introduction of `lua-placeholders`, LaTeX users have complete freedom in formatting invoice templates, and the invoice templates are directly integratable with the enhanced version of GinVoice.

Keywords

LuaLaTeX, YAML

1 Introduction

During my work as a software engineer, I encountered a challenge for a company that drafts agreements and terms for multiple clients. One of the challenging aspects was keeping client data and regulatory documentation separate. Previously, I addressed this challenge in GinVoice [3] by generating additional LaTeX files with Python, which were then compiled alongside the main LaTeX file. However, this time, my goal was to provide a solution from within the LaTeX domain itself, rather than the application domain. The solution I developed, now known as `lua-placeholders` [5], introduces a shared data layer with YAML between LaTeX and application code. The package provides an intermediary layer specifically for data through YAML files. To demonstrate this solution, we use GinVoice as an example. This example, a Python GTK application that generates invoices with LaTeX, offers slightly more complexity and challenges than the legal domain has to offer.

1.1 The compiler — LuaLaTeX

I decided to use LuaLaTeX as the compiler for several reasons. Since 2016, I have been using LuaLaTeX, which greatly helped me with documents within computer science at the time. Over the years, I have gained a lot of experience in compiling with LuaLaTeX and see it as a suitable compiler as a developer, thanks to the ability to script in Lua, which I naturally appreciate as a programmer.

The ability to script in Lua offers several advantages. It allows me to perform complex tasks during the compilation process, such as processing YAML files or manipulating and structuring data. Additionally, LuaLaTeX supports Lua init scripts, allowing me to implement a custom compilation process with its own command line interface (CLI), further simplifying and optimizing the integration process for end solutions.

1.2 What is YAML?

As a DevOps engineer, I have often encountered YAML while working with tools such as Docker Compose, Travis CI, GitHub Actions, and Canonical's NetPlan (Ubuntu systems). YAML is widely used in the DevOps world for automating and managing configurations, functioning as a structured markup language for defining configuration files and capturing infrastructural and operational aspects of software applications.

YAML has become a crucial component of modern software development and deployment due to its simple syntax and flexibility. In combination with LaTeX, YAML provides a powerful mechanism for defining and managing structured data, which is particularly useful when integrating client data into LaTeX documents. Listing 1 shows an example of YAML used in conjunction with LaTeX.

```
supplier: grapefruit
client: juicing-joker
title: Grapefruit Inc. Invoice
subtitle: for fruits and stuff
currency: \$
number: 1
date: \today
...
```

Listing 1: `invoice-001.yaml`

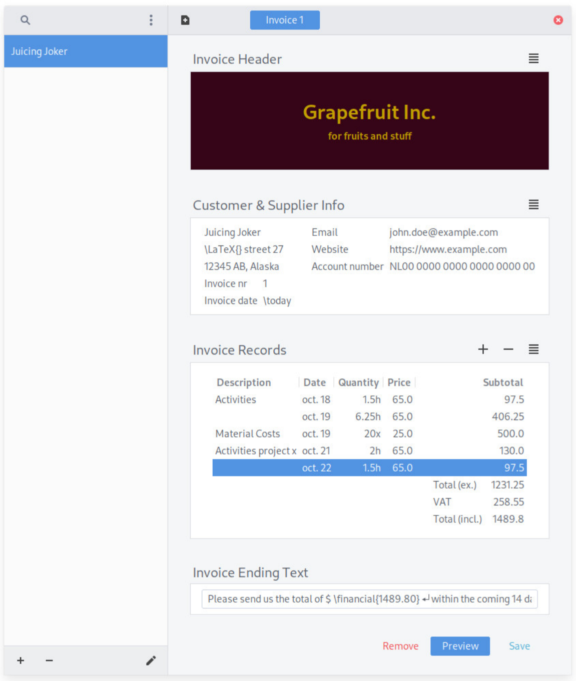
2 GinVoice

In this section, we will take a closer look at GinVoice, an open-source Python GTK application that utilizes LaTeX behind the scenes to create invoices. Additionally, we will examine the provided invoice template and delve into the associated data within the invoice.

2.1 The application

GinVoice has multiple views. The most common is the main view, where you can draft multiple invoices simultaneously. In this view, depicted in figure 1, almost all components are visible. You can see the header, information tables, invoice rules, and the closing text included in it. Figure 1 shows that the input fields are already filled in, and their content does not deviate much from the end result, as seen in figure 2. Other application views will be discussed later in this section.

doi.org/10.47397/tb/45-1/tb139nijenhuis-placeholders



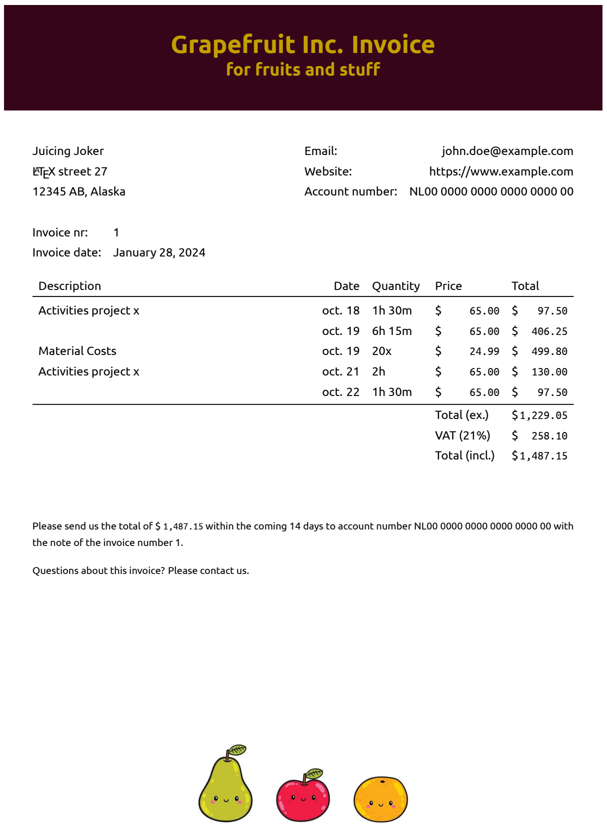
Grapefruit Inc.
for fruits and stuff

Juicing Joker		Email	john.doe@example.com
\LaTeX{} street 27		Website	https://www.example.com
12345 AB, Alaska		Account number	NL00 0000 0000 0000 0000 00
Invoice nr	1		
Invoice date	\today		

Description	Date	Quantity	Price		Subtotal
Activities	oct. 18	1.5h	65.0		97.5
	oct. 19	6.25h	65.0		406.25
Material Costs	oct. 19	20x	25.0		500.0
Activities project x	oct. 21	2h	65.0		130.0
	oct. 22	1.5h	65.0		97.5
				Total (ex.)	1231.25
				VAT	258.55
				Total (incl.)	1489.8

Please send us the total of $ \financial{1489.80} ↵ within the coming 14 d

Figure 1: GinVoice — the application



Grapefruit Inc. Invoice
for fruits and stuff

Juicing Joker		Email:	john.doe@example.com
LaTeX street 27		Website:	https://www.example.com
12345 AB, Alaska		Account number:	NL00 0000 0000 0000 0000 00
Invoice nr:	1		
Invoice date:	January 28, 2024		

Description	Date	Quantity	Price	Total
Activities project x	oct. 18	1h 30m	$ 65.00	$ 97.50
	oct. 19	6h 15m	$ 65.00	$ 406.25
Material Costs	oct. 19	20x	$ 24.99	$ 499.80
Activities project x	oct. 21	2h	$ 65.00	$ 130.00
	oct. 22	1h 30m	$ 65.00	$ 97.50
			Total (ex.)	$1,229.05
			VAT (21%)	$ 258.10
			Total (incl.)	$1,487.15

Please send us the total of $ 1,487.15 within the coming 14 days to account number NL00 0000 0000 0000 0000 00 with the note of the invoice number 1.

Questions about this invoice? Please contact us.

Figure 2: Sample invoice generated with GinVoice

2.2 LaTeX template

Below is an example of the code within the `document` environment:

```
\begin{document}
\thispagestyle{headermain}
\makeheader
\vspace{2cm}
\begin{tabular}{@{}l@{}}
    \begin{tabular}{@{}l@{}}
        \addressee
    \end{tabular} \\
    \begin{tabular}{@{}l l@{}}
        \customerinfo
    \end{tabular}
\end{tabular}
\hfill
\begin{tabular}{@{}l r@{}}
    \supplierinfo
\end{tabular}\\

\input{table}
\begin{invoice}{\columndef}{\tableheader}
               {\tablefooter}
    \tablerecords
\end{invoice}

{\footnotesize \theending{}}
\vfill
\begin{center}
    \images
\end{center}

\end{document}
```

Listing 2: `invoice.tex`

The source code in listing 2 demonstrates various macros that will be replaced by `lua-placeholders`: `\addressee`, `\customerinfo`, `\supplierinfo`, `\tablefooter`, `\tablerecords`, `\theending`, and `\images`. Additionally, there are variables such as title- and style-related information and `\currency` that will be handled.

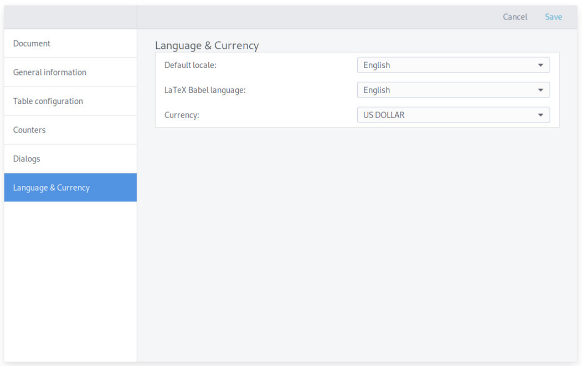
2.3 Generated LaTeX files

It is important to note that GinVoice [3] currently uses a Python script, `generator.py`, to generate additional TeX files. These TeX files are then included in the template using `\include`, making the necessary macros available.

Starting with the language setting:

```
\usepackage[english]{babel}
```

Listing 3: `languages.tex`



Language & Currency

Default locale:	English
LaTeX Babel language:	English
Currency:	US DOLLAR

Figure 3: Language settings

At the time, I chose to include a separate language setting in the application, as shown in figure 3, so that words within the invoice are correctly hyphenated using `babel`.

Another aspect within the preamble is setting the document properties. These macros are imported from the generated file `meta.tex`, whose macros are later used in the `\hypersetup`.

```
\global\def\currency{\$}
\global\def\author{Erik Nijenhuis}
\global\def\title{Grapefruit Inc. Invoice}
\global\def\subject{Invoice for Juicing Joker}
\global\def\keywords{Invoice Grapefruit ↩
  Juicing Joker}
\global\def\producer{GinVoice Generator}
\global\def\creator{gingen}
\global\def\continuationheader{\title{} -- ↩
  \subject{}}
\global\def\continuationfooter{See next page.}
```

Listing 4: `meta.tex`

Common macros, such as `\title`, are used in multiple places. That is also why the `\title` does not need to be in the `header.tex`.

```
\global\def\subtitle{for fruits and stuff}
```

Listing 5: `header.tex`

The customer's address is placed in a macro, with the address lines separated by a newline.

```
\newcommand{\addressee}{Juicing Joker\\ ↩
  \LaTeX{} street 27\\12345 AB, Alaska}
```

Listing 6: `addressee.tex`

This approach would be suitable for a table with a single column or for, say, an `enumerate` environment.

The customer and supplier information assumes a table environment with two columns.

```
\newcommand{\customerinfo}{
    & \\
    Invoice nr: & 1 \\
    Invoice date: & \today \\
}
```

Listing 7: `customer_info.tex`

```
\newcommand{\supplierinfo}{
    Email: & john.doe@example.com \\
    Website: & https://www.example.com \\
    Account number: & NL00 0000 0000 0000 ↩
      0000 00 \\
    & \\
    & \\
    & \\
}
```

Listing 8: `supplier_info.tex`

The drawback of this setup is that an ampersand (&) does not have any function within the context of the macro itself. That would only be the case when working within a `tabular` environment. Despite most LaTeX editors giving an error for this, strangely enough, this approach still works.

The most significant challenge within the application was making the invoice table configurable. For this, there is a separate view, as seen in figure 4. In the figure, you can see that each column can have a different width, including length of text, maximum available space, or hidden. This added complexity from the application resulted in quite complex output in the generated `table.tex` file, as shown in the following code:

```
\newlength{\rowsize}
\setlength{\rowsize}{\linewidth}
\newlength{\cIsize}
\settowidth{\cIsize}{oct. 22}
\addtolength{\rowsize}{-\cIsize}
\addtolength{\rowsize}{-2\tabcolsep}
```

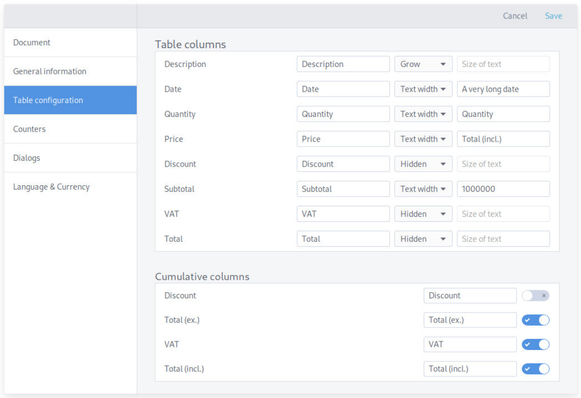


Table columns

Description	Description	Grow	Size of text
Date	Date	Text width	A very long date
Quantity	Quantity	Text width	Quantity
Price	Price	Text width	Total (incl.)
Discount	Discount	Hidden	Size of text
Subtotal	Subtotal	Text width	1000000
VAT	VAT	Hidden	Size of text
Total	Total	Hidden	Size of text

Cumulative columns

Discount	Discount	✗
Total (ex.)	Total (ex.)	✓
VAT	VAT	✓
Total (incl.)	Total (incl.)	✓

Figure 4: Table settings

```
\newlength{\cIIsize}
\settowidth{\cIIsize}{\textbf{Quantity}}
\addtolength{\rowsize}{-\cIIsize}
\addtolength{\rowsize}{-2\tabcolsep}
\newlength{\cIIIsize}
\settowidth{\cIIIsize}{\textbf{Total (incl.)}}
\addtolength{\rowsize}{-\cIIIsize}
\addtolength{\rowsize}{-2\tabcolsep}
\newlength{\cIVsize}
\settowidth{\cIVsize}{\$ 1,000.00}
\addtolength{\rowsize}{-\cIVsize}
\addtolength{\rowsize}{-2\tabcolsep}
\newcommand{\columncount}{5}
\newcolumntype\columndef ↩
  {L{1.00\rowsize-2\tabcolsep} R{\cIsize} ↩
  L{\cIIsize} F{\cIIIsize} F{\cIVsize}}
\newcommand{\tableheader}{\rowheadercolor ↩
  Description&\rowheadercolor ↩
  Date&\rowheadercolor ↩
  Quantity&\rowheadercolor ↩
  Price&\rowheadercolor Total\\}
\newcommand{\tablerecords}{
    Activities project x & oct. 18 & 1h 30m ↩
      & \currency\hfill\financial{65.00} & ↩
      \currency\hfill\financial{97.50}\\
     & oct. 19 & 6h 15m & ↩
       \currency\hfill\financial{65.00} & ↩
       \currency\hfill\financial{406.25}\\
    Material Costs & oct. 19 & 20x & ↩
      \currency\hfill\financial{24.99} & ↩
      \currency\hfill\financial{499.80}\\
    Activities project x & oct. 21 & 2h & ↩
      \currency\hfill\financial{65.00} & ↩
      \currency\hfill\financial{130.00}\\
     & oct. 22 & 1h 30m & ↩
       \currency\hfill\financial{65.00} & ↩
       \currency\hfill\financial{97.50}\\}
\newcommand{\cumoffset}{& & & }
\newcommand{\tablefooter}{\cum{Total ↩
  (ex.)}{1229.05}
\cum{VAT (21\%)}{258.10}
\cum{Total (incl.)}{1487.15}
}
```

Listing 9: `table.tex`

In addition to the complex column configuration, there are `\tablerecords` and `\tablefooter`, both similar to, for example, the supplier information.

The last generated file `footer.tex` defines the remaining missing macros, `\theending` and `\images`:

```
\newcommand{\theending}{Please send us the ↩
  total of \$ \financial{1487.15}
within the coming 14 days
to account number NL00 0000 0000 0000 0000 00
with the note of the invoice number 1.

Questions about this invoice?
Please contact us.}
\graphicspath{{/home/erik/share/ginvoice/img/}}
\newcommand{\images}{
    \includegraphics[width=.1\textwidth]{image1}
    \hspace{1.5em}
    \includegraphics[width=.1\textwidth]{image2}
    \hspace{1.5em}
    \includegraphics[width=.1\textwidth]{image3}
}
```

Listing 10: `footer.tex`

At the time, I chose to store all graphic files somewhere within the GinVoice environment. I linked this to LaTeX by using `\graphicspath`.

2.4 Invoice data

When looking at all the information coming from GinVoice, a few exceptions aside, we end up with the data presented in figure 5. For convenience, I have already divided all the information into separate entities, which will correspond to the YAML files, extensively discussed in the next section.

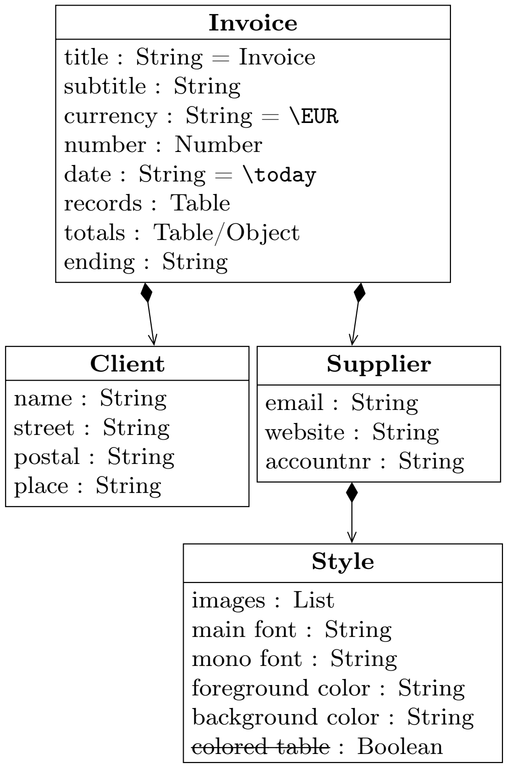


Figure 5: Class diagram of the invoice

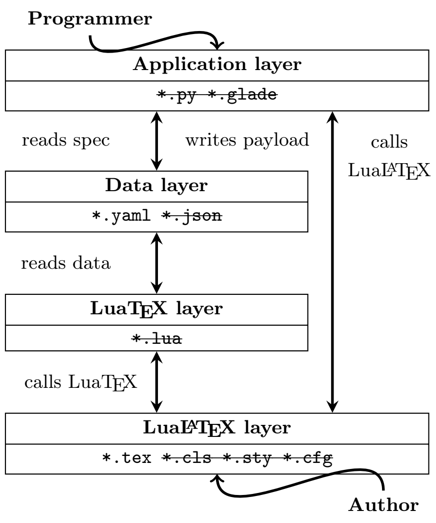


Figure 6: Levels within GinVoice

3 Invoice templates with `lua-placeholders`

This section demonstrates how YAML interfaces, also known as recipes, can be used as interfaces for invoice templates and how they can be linked to LaTeX.

The ultimate goal is to provide an efficient and customizable invoicing interface that can be easily integrated into an enhanced version of GinVoice. Figure 6 illustrates a representation of the new situation, with techniques irrelevant for this article crossed out.

Thus, the data, as seen in figure 5, is moved from the application level to the data level. This allows both Python programmers and LaTeX users to interact with the data level, something that is impossible in the current situation.

3.1 YAML specifications

Based on the data analysis in section 2.4, we can start working with the recipes. All recipes are placed in the `recipes` directory relative to the LaTeX project. Alternatively, you could store the `recipes` directory under `$TEXMFHOME/tex/` to make the recipes available everywhere.

3.1.1 The invoice

The invoice recipe, `recipes/invoice.yaml`, specifies two relationships: `supplier` and `client`, as mentioned earlier in section 2.4.

```yaml
# Actors
supplier:
  type: string
client:
  type: string
```

Listing 11: `recipes/invoice.yaml`

How the corresponding recipes are loaded based on these values is described in section 3.2.3.

The data within the invoice part can optionally be standardized using a `default` field, as done for `title`. You can even invoke LaTeX from a default value, including other parameters using `\param`.

```yaml
# Invoice variables
title:
  type: string
  default: Invoice \param{number}
subtitle:
  type: string
  placeholder: Subtitle
currency:
  type: string
  default: \EUR
number:
  type: string
  placeholder: Invoice number
date:
  type: string
  placeholder: Invoice date
```

In addition to default values, temporary placeholders can also be specified.

The most complex part of the invoice is the invoice table, where you can specify columns just like you do for other data types.

```yaml
records:
  type: table
  columns:
    description:
      type: string
    date:
      type: string
    quantity:
      type: string
      default: 0x
    price:
      type: number
      default: 0
    total:
      type: number
      default: 0
```

For most LaTeX users, the `total` column can be omitted and calculated using a package like `invoice2` [2]. To do that, it is also necessary to make the `quantity` field of type `number` and add an extra field like `quantity type`, so that you can display the correct notation for the `quantity` column.

For the final totals, I chose the type `object` so that I can manually set the different totals in LaTeX.

```
totals:
  type: object
  fields:
    total ex:
      type: number
      default: 0
    vat:
      type: number
      default: 0
    total incl:
      type: number
      default: 0
```

The final totals could also be handled in a more generic way, like the `extra fields` field in the supplier recipe (see section 3.1.3).

The last field of the invoice, `message`, uses the special YAML feature of multiline strings in the default value.

```
message:
  type: string
  default: |
    Please send us the total of ↩
      \currency~\paramfield{totals}{total ↩
      incl}
    within the coming 14 days to account ↩
      number
    \param[supplier]{account number} with ↩
      the note of the invoice number ↩
      \param{number}.\\[2em]

    Questions about this invoice? Please ↩
      contact us.
```

Using the pipe (`|`) activates this mode. This construction is ideal for large texts, possibly with LaTeX syntax.

3.1.2 Client

The client data does not have any special specifications compared to the invoice.

```
name:
  type: string
  placeholder: Client namme
street:
  type: string
  placeholder: Street + nr
postal:
  type: string
  placeholder: 9999 ZZ
place:
  type: string
  placeholder: City
```

Listing 12: `recipes/client.yaml`

Alternatively, all address details could be specified as a `list` type, along with a specification, as seen in `extra fields` in the supplier recipe. This would make the interface more generic but less adaptable within the LaTeX context.

3.1.3 Supplier

In the recipe for the supplier, the `style` field serves the same function as `supplier` and `client` of the invoice, allowing the user to choose which style to apply.

```
name:
  type: string
  placeholder: Supplier name
email:
  type: string
  placeholder: Email
website:
  type: string
account number:
  type: string
  placeholder: Account number
extra fields:
  type: table
  columns:
    key:
      type: string
    val:
      type: string
# Suppliers style
style:
  type: string
```

Listing 13: `recipes/supplier.yaml`

Another interesting field in this specification is `extra fields`. This field uses the `table` type to allow arbitrary additional information fields, such as the supplier's account number, VAT number, or any other relevant details. Using a table instead of a fixed number of fields gives the end-user the flexibility to add as much extra information as needed, without imposing restrictions.

3.1.4 Style

In the style recipe, fonts, colors, and multiple images can be specified. As mentioned earlier: for LaTeX users, this could be fully specified in LaTeX itself. The style recipe could then be omitted.

```
images:
  type: list
  item type: string
main font:
  type: string
  default: Ubuntu
mono font:
  type: string
  default: Ubuntu Mono
foreground color:
  type: string
  default: 000000
background color:
  type: string
  default: FFFFFF
```

Listing 14: `recipes/style.yaml`

A notable point here is the type for `images`, namely `list`. In section 3.3, you can see how this list is loaded at the bottom of the invoice.

3.2 The new invoice

Now that the recipes are in order, we can proceed to integrate them into LaTeX (in `invoice.tex`).

3.2.1 Loading recipes in the preamble

The recipes are loaded using the `\loadrecipe` macro.

```
\loadrecipe[\jobname]{recipes/invoice.yaml}
\loadrecipe{recipes/supplier.yaml}
\loadrecipe{recipes/client.yaml}
\loadrecipe{recipes/style.yaml}
```

For the `invoice` recipe, you can see that it is given a ⟨*namespace*⟩ of `\jobname` (the optional argument). This is because the `\param` macro by default uses `\jobname` as the ⟨*namespace*⟩, simplifying its use.

The other recipes do not specify a ⟨*namespace*⟩, meaning they use the 'basename' of the path as the ⟨*namespace*⟩. In this case, respectively, `supplier`, `client`, and `style`.

3.2.2 Currency

Regarding the currency, I have chosen to disguise it in the `\currency` macro. This is because it is also used in other files, such as `invoice.cls`.

```
\def\currency{\rawparam{\jobname}{currency}}
```

If the ⟨*currency*⟩ is not set, the default value from `style.yaml` is used. In this case, it defaults to `\EUR`.

3.2.3 Loading values

I've chosen to manage all YAML files related to the data in corresponding directories.

```
⟨project name⟩
├─ recipes
│  └─ ⟨recipe⟩.yaml
├─ invoices
│  └─ ⟨invoice-xxx⟩.yaml
├─ clients
└─ etc.
```

Values, also called the payload, are loaded similarly to recipes but with the `\loadpayload` macro. Due to the relationships described in section 2.4, it is slightly more complex than recipes because `lua-placeholders` does not offer anything standard for this.

```
\IfFileExists{invoices/\jobname.yaml}{
    \loadpayload[\jobname] ↩
      {invoices/\jobname.yaml}
    \strictparams
}{}
```

When loading invoice values, it is checked whether a corresponding YAML file exists. If so, that payload is loaded, and the experimental macro `\strictparams` is used, which means that errors will occur in the future if mandatory data is missing. If no corresponding file is found, an invoice template is compiled.

After loading the invoice data, we can check if a client is specified in the invoice data. We do this using `\hasparam`. This concerns the invoice data, for which we do not need to specify a ⟨*namespace*⟩.

```
\hasparam{client}{%
    \loadpayload[client] ↩
      {clients/\rawparam{\jobname} ↩
      {client}.yaml}
}{}
```

Generally, `\param` is not intended for use within the preamble because it can also yield placeholders with LaTeX markup. For such difficult situations, the macro `\rawparam` is written, as done for the client and supplier. This macro has no optional arguments; they often cause problems with, for example, `pgfkeys`.

```
\hasparam{supplier}{%
    \loadpayload[supplier] ↩
      {suppliers/\rawparam{\jobname} ↩
      {supplier}.yaml}
}{}
```

As you can see, loading the supplier does not differ from loading the client. However, there is a follow-up action after loading the supplier, namely checking if the style can be loaded. This is done in the same way as with the client and supplier themselves, but here you see that the ⟨*namespace*⟩ must be set.

```
\hasparam[supplier]{style}{%
    \loadpayload[style] ↩
      {styles/\rawparam{supplier}{style}.yaml}
    \setmainfont{\rawparam{style}{main ↩
      font}}
    \setmonofont{\rawparam{style}{mono ↩
      font}}
68     \definecolor{backgroundcolor}{HTML} ↩
         {\rawparam{style}{background color}}
69     \colorlet{bgcolor}{backgroundcolor}
70     \definecolor{foregroundcolor}{HTML} ↩
         {\rawparam{style}{foreground color}}
71     \colorlet{textcolor}{foregroundcolor}
72 }{}
```

For the style-related data, I chose to configure the values directly in the corresponding macros, such as `\setmainfont` and `\definecolor`, as long as a style is specified. You could also choose to set the style values by default based on the default values specified in the `style` recipe, by placing the configuration outside the `\hasparam` block.

3.3 Processing in the document

Before we can move on to compiling invoices, we have one more task: setting all values in the document itself.

3.3.1 Header

The `\makeheader` macro comes from `invoice.cls`. It expects the title and subtitle as arguments, for which we use `\param`:

```
76 \begin{document}
77 \thispagestyle{headermain}
78 \makeheader{\param{title}}{\param{subtitle}}
79 \vspace{2cm}
```

3.3.2 Information

The left column of the information is quite tricky, as it contains both client information and invoice data, such as the number and date.

```
80 \begin{tabular}{@{}l@{}}
81     \begin{tabular}{@{}l@{}}
82         \param[client]{name}\\
83         \param[client]{street}\\
84         \param[client]{postal}, ↩
             \param[client]{place}\\
85     \end{tabular} \\
86     \begin{tabular}{@{}l l@{}}
87         Invoice number: & \param{number}\\
88         Invoice date: & \param{date}\\
89     \end{tabular}
90 \end{tabular}
91 \hfill
```

You can see in the address lines that a line break is set for each line. This could also have been done if, for example, a field `address lines` of type `list` was present. Then it would have been solved in one go with `\param[client]{address lines}`, assuming that `postal` and `place` are merged on one line in YAML. This alternative assumes that the `\paramlistconjunction` macro is set to '`\\`', instead of the default '`,~`'.

```
92 \begin{tabular}{@{}l r@{}}
93     Company: & \param[supplier]{name} \\
94     Email: & \param[supplier]{email} \\
95     Website: & \param[supplier]{website} \\
96     Account nr: & ↩
         \param[supplier]{account number} \\
97     \hasparam[suplier]{extra fields}{%
98         \def\formatsupplierextra{\key & ↩
             \val\\}%
99         \fortablerow[supplier]{extra ↩
             fields}{formatsupplierextra}
100    }{}
101 \end{tabular}\\
```

The right column of information is similar to the left, except it has one additional special field, namely `extra fields` of type `table`. This allows for a variable number of rows to be added. The same could potentially be applied to the client details in the left column. Then only the choice remains whether to place them above or below the invoice information.

3.3.3 Table

As mentioned earlier, standardizing the column definition is difficult.

On line 105, you can see what the `\columdefs` could have provided, except for the counters that I previously used.

```
103 \begin{invoice}
104 % Column definition based on 540pt
105 {@{}L{180pt-\tabcolsep} ↩
      R{80pt-2\tabcolsep} ↩
      L{60pt-2\tabcolsep} ↩
      F{120pt-2\tabcolsep} ↩
      F{100pt-\tabcolsep}@{}}
```

For the second argument of the `invoice` environment, a static header is set.

```
106 % Header
107 {\textbf{Description} & \textbf{Date} & ↩
      \textbf{Quantity} & \textbf{Price} & ↩
      \textbf{Total} \\ \hline}
```

For the third argument of the `invoice` environment, you can see how the final totals are set in the table. These totals are placed in the last two columns of each row, so that they align neatly with the rest of the table.

```
% Totals
{%
    & & & \textbf{Total (ex.)} & \currency\hfill{\ttfamily \paramfield{totals}{total ex}} \\
    & & & \textbf{VAT} & \currency\hfill{\ttfamily \paramfield{totals}{vat}} \\
    & & & \textbf{Total (incl.)} & \currency\hfill{\ttfamily \paramfield{totals}{total incl}} \\
}
```

In the final part of the table, you can see how each invoice line is set using `\fortablerow` with the help of `\formatrecords`.

```
    \newcommand\formatrecords{%
        \description & \date & \quantity &%
        \currency\hfill{\ttfamily\price} &%
        \currency\hfill{\ttfamily\total} \\}
    \fortablerow{records}{formatrecords}
\end{invoice}
```

The overall structure of the table is still from the previous situation. The notable difference from the old situation is that the data can be put into any sort of table structure, since the data is decoupled from the LaTeX and application domains, and the challenges of typesetting are shifted to the LaTeX domain.

3.3.4 Closing text and images

Where we previously saw an advanced YAML specification for the `message` field, the implementation in LaTeX remains virtually the same:

```
{\footnotesize\param{message}}
```

The only difference is:
`\theending` $\rightarrow$ `\param{message}`

The images, on the other hand, are slightly more difficult to implement in LaTeX due to the `list` type.

```
\newcommand\formatimage[1] {\hspace{.75em}\includegraphics [width=2cm]{#1}\hspace{.75em}}%
\hasparam[style]{images}{%
    \vfill
    \begin{center}
        \forlistitem[style]{images} {formatimage}
    \end{center}
}{}
\end{document}
```

Where previously in Python all images were neatly placed next to each other, with a `\hspace` of `1.5em` between each image, I chose to insert half that value as an `\hspace` on each side of each image. This is because the `\forlistitem` macro does not yet have a convenient way to specify a separator, like `\param` does by setting `\paramlistconjunction` to '`\hspace{1.5em}`'.

4 Execution

Now that the legacy invoice has been completely transformed, let's see what the result looks like. If you want to participate via the command line, please refer to the full source code [4] of these examples.

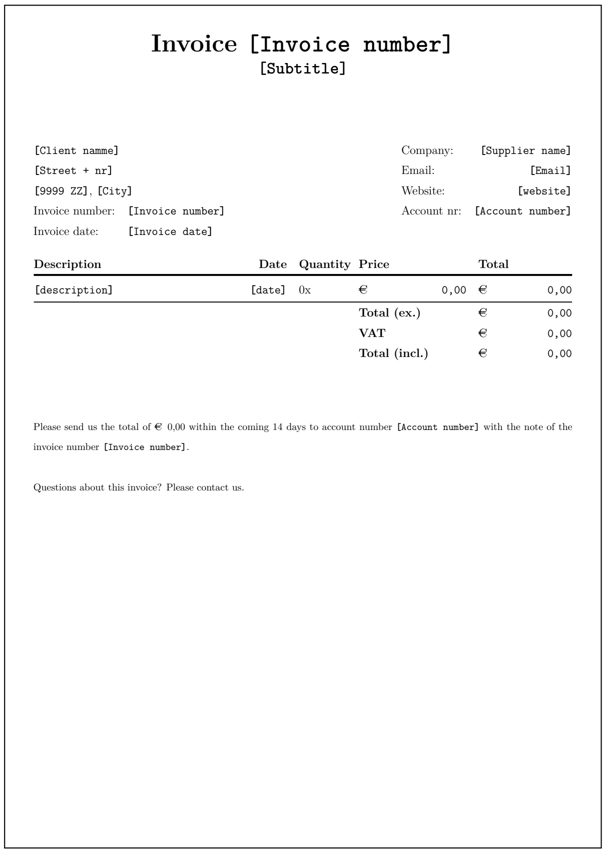
4.1 The template version

Without providing any values, we get the following result, as shown in figure 7.

As mentioned earlier, `lua-placeholders` can only be compiled with LuaLaTeX. The example can be compiled as follows:

```
lualatex --jobname=invoice-template \
    --output-directory="${OUTPUT_DIR}" \
    invoice
```

Listing 15: Compiling with `lualatex`



Invoice [Invoice number]
[Subtitle]

[Client namme]		Company:	[Supplier name]
[Street + nr]		Email:	[Email]
[9999 ZZ], [City]		Website:	[website]
Invoice number:	[Invoice number]	Account nr:	[Account number]
Invoice date:	[Invoice date]		

Description	Date	Quantity	Price		Total	
[description]	[date]	0x	€	0,00	€	0,00
			Total (ex.)		€	0,00
			VAT		€	0,00
			Total (incl.)		€	0,00

Please send us the total of € 0,00 within the coming 14 days to account number [Account number] with the note of the invoice number [Invoice number].

Questions about this invoice? Please contact us.

Figure 7: `invoice-template.pdf`

where `${OUTPUT_DIR}` is the desired output directory.

However, if you are designing a template, continuous generation with `latexmk` [1] is more user-friendly:

```
latexmk -pvc -lualatex \
    --jobname=invoice-template \
    --output-directory="${OUTPUT_DIR}" \
    invoice
```

Listing 16: Compiling with `latexmk`

With the `-pvc` option, you don't have to recompile with TEX every time there is a change; it happens automatically.

4.2 YAML values

To get a filled invoice, we will need the following YAML files:

```
⟨project dir⟩
├── invoices
│   └── ⟨invoice⟩.yaml
├── suppliers
│   └── ⟨supplier⟩.yaml
├── styles
│   └── ⟨style⟩.yaml
└── clients
    └── ⟨client⟩.yaml
```

This structure is based on the implementation described in section 3.2.3. Before discussing the contents of the YAML files, let's first consider alternative project structures.

4.2.1 Alternative project structure

Everyone is free to create their desired folder structure. For example, you could place styles under

`/suppliers/`⟨*supplier*⟩`/style.yaml`

so that you can even omit the `style` field in the supplier recipe. Another option is to place the `clients` folder under the supplier level, so you don't accidentally mix clients of different suppliers. This could be achieved as follows:

```
⟨project dir⟩
└── suppliers
    ├── ⟨supplier⟩.yaml
    └── ⟨supplier⟩
        └── ⟨client⟩.yaml
```

This way, the implementation for loading clients would require the variables ⟨*supplier*⟩ and ⟨*client*⟩, to then reach the path

`suppliers/`⟨*supplier*⟩`/`⟨*client*⟩`.yaml`.

The same consideration could be applied to the invoices, but this is a more difficult scenario, as the invoice data is based on `\jobname` in the implementation of section 3.2.3. One possible solution for this is to manage the project per supplier. You can then place the *recipes* in the `$TEXMFHOME/tex` directory so that they are available for all projects. Here's an example of a possible project structure:

```
$HOME/texmf/tex                 ⟨project dir⟩
└── recipes                     ├── invoices
│   ├── invoice.yaml            │   └── ⟨invoice⟩.yaml
│   ├── client.yaml             ├── clients
│   ├── supplier.yaml           │   └── ⟨client⟩.yaml
│   └── style.yaml              ├── supplier.yaml
├── invoice.cls                 └── style.yaml
└── invoice.tex
```

In this example, all data is separated per supplier, including client information and final invoices.

4.3 Suppliers and clients

In the example result of GinVoice, a client *Juicing Joker* was shown. In YAML, this would translate to:

```
name: Juicing Joker
street: \LaTeX-street 27
postal: 12345 AB
place: Alaska
```

Listing 17: `clients/juicing-joker.yaml`

This way, the client can be referenced in the invoice with `juicing-joker`.

For the supplier, we saw *Grapefruit Inc.* in the example, which translates to:

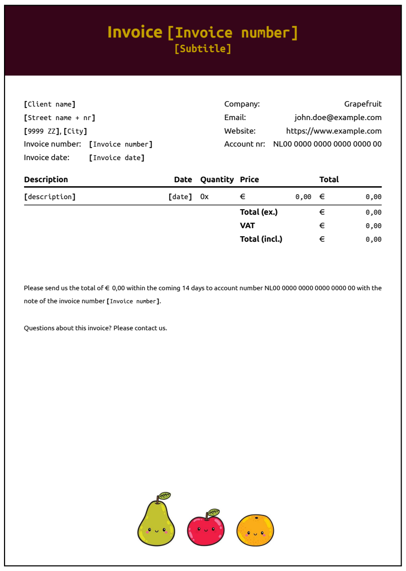
```
name: Grapefruit
email: john.doe@example.com
website: https://www.example.com
account number: NL00 0000 0000 0000 0000 00
style: grapefruit
```

Listing 18: `suppliers/grapefruit.yaml` or `grapefruit/supplier.yaml`

And for the style:

```
main font: Ubuntu
mono font: Ubuntu Mono
foreground color: c4a000
background color: 360519
images:
  - img/image1
  - img/image2
  - img/image3
```

Listing 19: `styles/grapefruit.yaml` or `grapefruit/style.yaml`



Invoice [Invoice number]
[Subtitle]

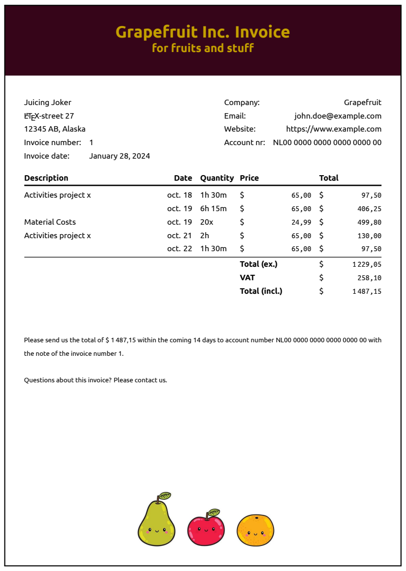
[Client name]		Company:	Grapefruit
[Street name + nr]		Email:	john.doe@example.com
[9999 ZZ], [City]		Website:	https://www.example.com
Invoice number:	[Invoice number]	Account nr:	NL00 0000 0000 0000 0000 00
Invoice date:	[Invoice date]		

Description	Date	Quantity	Price		Total	
[description]	[date]	0x	€	0,00	€	0,00
			Total (ex.)		€	0,00
			VAT		€	0,00
			Total (incl.)		€	0,00

Please send us the total of € 0,00 within the coming 14 days to account number NL00 0000 0000 0000 0000 00 with the note of the invoice number [Invoice number].

Questions about this invoice? Please contact us.

(a) `invoice-template.pdf`



Grapefruit Inc. Invoice
for fruits and stuff

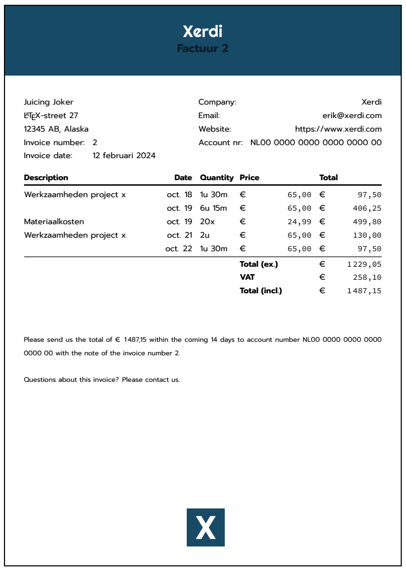
Juicing Joker		Company:	Grapefruit
LaTeX-street 27		Email:	john.doe@example.com
12345 AB, Alaska		Website:	https://www.example.com
Invoice number:	1	Account nr:	NL00 0000 0000 0000 0000 00
Invoice date:	January 28, 2024		

Description	Date	Quantity	Price		Total	
Activities project x	oct. 18	1h 30m	$	65,00	$	97,50
	oct. 19	6h 15m	$	65,00	$	406,25
Material Costs	oct. 19	20x	$	24,99	$	499,80
Activities project x	oct. 21	2h	$	65,00	$	130,00
	oct. 22	1h 30m	$	65,00	$	97,50
			Total (ex.)		$	1 229,05
			VAT		$	258,10
			Total (incl.)		$	1 487,15

Please send us the total of $ 1 487,15 within the coming 14 days to account number NL00 0000 0000 0000 0000 00 with the note of the invoice number 1.

Questions about this invoice? Please contact us.

(b) `invoice-001.pdf`



Xerdi
Factuur 2

Juicing Joker		Company:	Xerdi
LaTeX-street 27		Email:	erik@xerdi.com
12345 AB, Alaska		Website:	https://www.xerdi.com
Invoice number:	2	Account nr:	NL00 0000 0000 0000 0000 00
Invoice date:	12 februari 2024		

Description	Date	Quantity	Price		Total	
Werkzaamheden project x	oct. 18	1u 30m	€	65,00	€	97,50
	oct. 19	6u 15m	€	65,00	€	406,25
Materiaalkosten	oct. 19	20x	€	24,99	€	499,80
Werkzaamheden project x	oct. 21	2u	€	65,00	€	130,00
	oct. 22	1u 30m	€	65,00	€	97,50
			Total (ex.)		€	1 229,05
			VAT		€	258,10
			Total (incl.)		€	1 487,15

Please send us the total of € 1487,15 within the coming 14 days to account number NL00 0000 0000 0000 0000 00 with the note of the invoice number 2.

Questions about this invoice? Please contact us.

(c) `invoice-002.pdf`

Figure 8: Invoice Examples

The advantage of the alternative project structure is that `invoice-template` automatically picks up the styling as well as the supplier information, as seen in figure 8a.

4.4 Invoices

To create an invoice that exactly matches the standard example of GinVoice, as seen in figure 8b, we use the following YAML example:

```yaml
supplier: grapefruit
client: juicing-joker
title: Grapefruit Inc. Invoice
subtitle: for fruits and stuff
currency: \$
number: 1
date: January 28, 2024
records:
  - description: Activities project x
    date: oct. 18
    quantity: 1h 30m
    price: 65
    total: 97.5
  - description: ''
    date: oct. 19
```

Listing 20: `invoices/invoice-001.yaml`

The actors `grapefruit` and `juicing-joker`, discussed in section 4.3, are seen in the invoice. Additionally, the example has the same general information to achieve the same result. In the `records` field, you can see that one row of the table takes up many lines. In the second row of the table, you can see that the `description` field has an empty value. If the quotes are omitted in YAML, you will get an error when converting to data. Since the rows do not differ too much from each other, we continue the example at the `totals` field:

```yaml
totals:
  total ex: 1229.05
  vat: 258.10
  total incl: 1487.15
message: |
  Please send us the total of \currency~\paramfield{totals}{total incl}
  within the coming 14 days to account number
  \param[supplier]{account number} with the note of the invoice number \param{number}.\\[2em]

  Questions about this invoice? Please contact us.
```

Lastly in the example, we see the totals and the closing text.

This invoice can then be compiled with the following command:

```
lualatex --jobname=invoice-001 \
    --output-directory="${OUTPUT_DIR}" \
    invoice
```

5 Conclusion

In this study, we have not only examined the implementation of invoice templates in GinVoice but also proposed an innovative method to seamlessly integrate these templates with the LaTeX ecosystem. By using YAML as an intermediate layer and `lua-placeholders` for dynamic insertions, we have provided a robust and flexible solution for invoice generation while creating a framework where various document components, such as client information, can be utilized across documents.

This approach not only grants LaTeX users the freedom to customize invoice templates as desired but also opens the door to a wider range of applications. By employing the same YAML-based structure, different documents, including contracts and invoices, can be generated and maintained with ease. This not only enhances consistency across various document types but also boosts the efficiency of the documentation process as a whole.

The utilization of `lua-placeholders` in conjunction with YAML enables the addition of dynamic content to templates, resulting in a more streamlined workflow for users. This flexibility makes it easy to separate data and formatting across different documents while allowing these components to be used across documents.

In conclusion, this approach not only makes a valuable contribution to optimizing billing processes but also unveils new possibilities for efficiently generating and managing various types of documents within an organization.

6 Discussion

6.1 LaTeX compilers

In the article, I assume the LuaLaTeX compiler. For other compilers, `lua-placeholders` does not provide a solution. Although some compilers still offer support for Lua, `lua-placeholders` does not take this into account. Research and implementation could improve the adoption of `lua-placeholders` within the LaTeX community.

6.2 JSON vs. YAML

I did not delve into the choice of YAML over JSON in the article. Both are intended for data, and while JSON is more well-known and has broader compatibility with programming languages, I chose YAML for the sake of readability of LaTeX source code. As demonstrated extensively, the files contain a lot of LaTeX source code. When using JSON every backslash would need to be escaped. For example:

```
title: Invoice \param{number}
```

Listing 21: YAML example

```
{"title": "\\param{number}" }
```

Listing 22: JSON example

As a LaTeX user, I find it more convenient to adjust values in YAML for testing purposes than in JSON.

6.3 GinVoice roadmap

Development has been stagnant for some time, but I recently discovered that the solution can also work for Windows platforms. Bringing GinVoice to the Windows platform significantly expands the target audience and, in my expectation, could garner more support for LaTeX.

As for the introduction of `lua-placeholders`, there are still a few obstacles to overcome, such as challenges related to translation and the variable column definition, which is precisely a user-friendly part of the application that has not been discussed.

References

[1] J. Collins, E. McLean, D.J. Musliner. *The latexmk package.* `www.cantab.net/users/johncollins/latexmk/index.html`

[2] S. Dierl. *The invoice2 package.* `github.com/no-preserve-root/invoice2`

[3] E. Nijenhuis. *The GinVoice GTK application.* `gitlab.gnome.org/MacLotsen/ginvoice`

[4] E. Nijenhuis. *The GinVoice template.* `github.com/Xerdi/ginvoice-template/tree/maps`

[5] E. Nijenhuis. *The lua-placeholders package.* `ctan.org/pkg/lua-placeholders`

◇ Erik Nijenhuis
Frans Halsstraat 38
Leeuwarden, 8932 JC
The Netherlands
`erik (at) xerdi dot com`
`https://github.com/MacLotsen`

Building a modern editing environment on Windows around GNU Emacs and AUCTEX

Arash Esbati

Abstract

In this article, we describe how to set up GNU Emacs with AUCTEX as an editing environment for (LA)TEX on Microsoft Windows using the MSYS2 distribution.

1 Introduction

What we know today as GNU Emacs is a text editor originally developed by Richard Stallman, who is also the founder of the Free Software Foundation (FSF) and the initiator of the GNU Project. The earliest recorded release of GNU Emacs is version 13 from March 1985, though a long history preceded that. Emacs' original inception was as a set of macros and keybindings for the TECO text editor (hence the meaning of "Emacs" as "editor macros"). For a thorough overview of Emacs timeline and technical development, please refer to [4].

TEX had major releases TEX78, TEX82 and TEX3.0 in 1990. Considering that these programs were developed more or less in the same time period and both are free software (free in terms of "free software" and not "open source"[6]), it is not a surprise that Emacs has a long history and very good support for editing TEX files.

Software around TEX and Emacs have their heritage on Unix-like operating systems where the source is provided and the software is built by distros or by the user. On Microsoft Windows, the process of building software by a user is rather uncommon. Finally, porting and building *nix software on Windows is not a task for casual users.

And this is where MSYS2 comes into play. It introduces itself as "a collection of tools and libraries providing an easy-to-use environment for building, installing and running native Windows software. ... Our package repository contains more than 2.600 pre-built packages ready to install." [5]

We will use MSYS2 in order to compile Emacs from the source and install auxiliary packages which will be used by Emacs during editing.

2 Installing the MSYS2 distribution

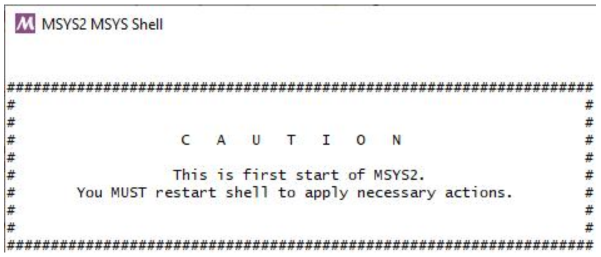
Before we start: As of March 2020, MSYS2 cannot be installed on a 32-bit system. And since January 2023, MSYS2 no longer supports Windows 7 and 8.0. So we need a 64-bit version of Windows 8.1 or higher in order to install the distribution. There are two other points to consider:

1. choosing a installation directory, and
2. choosing a `HOME` directory.

Regarding item 1, MSYS2 requires extracting its distribution into a folder where the name consists of only ASCII characters and no spaces. It also makes good sense to use a path that is not too long (due to `PATH_MAX` being 260); `c:\msys64` is ideal.

Regarding item 2, I suggest following the advice "ASCII only, no spaces", and possibly choosing a directory other than `c:\Users\<username>`. I recommend setting the value of the `HOME` environment variable to the directory chosen above globally on Windows — this is the only variable set outside MSYS2. The `HOME` directory is the place where Emacs looks for its init file upon start.[1]

Now we can fetch MSYS2 from `repo.msys2.org/distrib`. We want to install the portable version, so we download the `msys2-x86_64-latest.tar.xz` archive and unpack it under `c:\`. In the file Explorer, we go to `c:\msys64` and double click on `msys2.exe` which opens a MSYS shell, does the initial setup and ideally shows:



```
###################################################################
#                                                                 #
#                                                                 #
#                    C   A   U   T   I   O   N                    #
#                                                                 #
#                  This is first start of MSYS2.                  #
#       You MUST restart shell to apply necessary actions.        #
#                                                                 #
#                                                                 #
###################################################################
```

We follow the advice, close the window and double click `msys2.exe` again. To update all packages we run the command `pacman -Syu`. We follow the instructions and close the terminal if requested, then we start a new terminal and update again with `pacman -Su`. That's it!

Working with MSYS2 is easy: If we want to update the distribution or install new packages, we open a MSYS shell (`msys2.exe`) and when we want to use the installed packages, we open a MinGW64 shell (`mingw64.exe`).[2]

3 Installing Emacs

There are some options available for installing Emacs on Windows. The current stable release is Emacs 29.1.

3.1 Emacs release

The Emacs project provides pre-compiled binaries for Windows on a best-effort basis from `ftpmirror.gnu.`

[1] `gnu.org/software/emacs/manual/html_node/efaq-w32/Location-of-init-file.html`

[2] This is the way the author uses MSYS2; changing the shells isn't strictly necessary any more; it is more a habit.

org/emacs in the windows/ subdirectory where each major version of Emacs is kept in its own subdirectory. The compressed files also contain the libraries needed to support various features in Emacs, such as image support.

3.2 MSYS2 release

The MSYS2 project provides also pre-compiled Emacs binaries, usually the latest stable version. It can be installed via `pacman` with:

```
MSYS shell
$ pacman -S mingw-w64-x86_64-emacs
```

3.3 Building from the source

First, we have to install some tools we need for building Emacs. We want to have a full-fledged Emacs, hence we install a large number of packages. Please refer to [2] for more details. We run `msys2.exe` and enter the following command in the shell (you can paste them into the shell with `Shift+Ins`):

```
MSYS shell
$ pacman -S --needed base-devel \
mingw-w64-x86_64-toolchain      \
mingw-w64-x86_64-xpm-nox        \
mingw-w64-x86_64-gmp            \
mingw-w64-x86_64-giflib         \
mingw-w64-x86_64-gnutls         \
mingw-w64-x86_64-harfbuzz       \
mingw-w64-x86_64-jansson        \
mingw-w64-x86_64-lcms2          \
mingw-w64-x86_64-libjpeg-turbo  \
mingw-w64-x86_64-libpng         \
mingw-w64-x86_64-librsvg        \
mingw-w64-x86_64-libtiff        \
mingw-w64-x86_64-libwebp        \
mingw-w64-x86_64-libxml2        \
mingw-w64-x86_64-sqlite3        \
mingw-w64-x86_64-tree-sitter    \
mingw-w64-x86_64-zlib
```

We will build the current development version of Emacs from the Git repository. The code is in sync with what will be Emacs 30. First, install Git:

```
MSYS shell
$ pacman -S git
```

The `autocrlf` feature of Git may interfere with the configure file, so we disable it by running:

```
MSYS shell
$ git config --global core.autocrlf false
```

Next we close the current MSYS shell and run `mingw64.exe`. We clone the Emacs repository under a temporary directory:

```
MinGW64 shell
$ mkdir emacs-git
$ cd emacs-git
$ git clone \
https://git.savannah.gnu.org/git/emacs.git
$ cd emacs
```

The next series of commands builds Emacs.[3] With this setup, there is no need to install Emacs; we can invoke it out of the Git tree from the `src` directory. But to do an installation, we create a directory and run `make install` passing that directory to `prefix`:

```
MinGW64 shell
$ mkdir -p /c/msys64/opt/emacs
$ ./autogen.sh
$ ./configure --with-native-compilation \
--without-dbus --without-imagemagick \
--without-mailutils --without-pop
$ make
$ make install prefix=/c/msys64/opt/emacs
```

Note that we can run `make` with the `-j` option:

```
$ make -jN
```

where `N` is the number of CPU-cores in our system; the parallel execution will run significantly faster, speeding up the build process.

3.4 Adjusting the $PATH

The final step is to add the directory which contains `emacs.exe`, e.g., `c:\msys64\opt\emacs\bin`, to our `$PATH`. We do this in our `~/.bash_profile`:

```
MinGW64 shell
$ cd ~
$ touch .bash_profile
$ echo 'export \
PATH=$PATH:/c/msys64/opt/emacs/bin' \
>>.bash_profile
```

And while we're at it, we do the same for TEX Live:

```
MinGW64 shell
$ echo 'export \
PATH=$PATH:/c/texlive/2023/bin/windows' \
>>.bash_profile
```

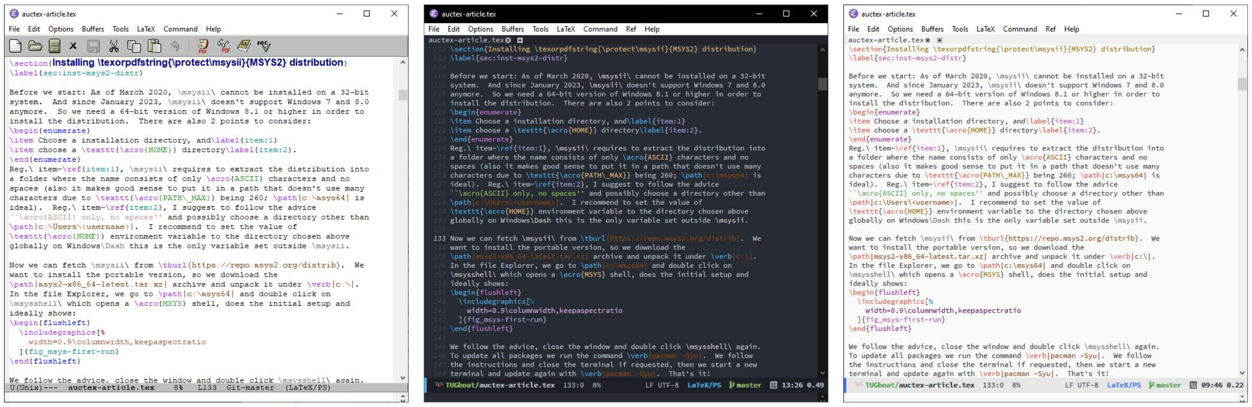
4 Starting Emacs

The above installs Emacs as a portable application. We will configure other applications, that we'll install later, in our `~/.bash_profile`. So we have to invoke Emacs and other programs from the command-line interface (CLI), `mingw64.exe` in our case, which starts `bash`. We type:

```
MinGW64 shell
$ emacs &
```

This starts Emacs in graphical mode, as shown in figure 1.

[3] There is a known issue with GCC 13.1; if the build process breaks, have a look at the file `etc/PROBLEMS` in the Emacs source tree and search for "Building the MS-Windows port with native compilation fails".



```
\section{Installing \texorpdfstring{\protect\msysii}{MSYS2} distribution}
\label{sec:inst-msys2-distr}

Before we start: As of March 2020, \msysii\ cannot be installed on a 32-bit
system.  And since January 2023, \msysii\ doesn't support Windows 7 and 8.0
anymore.  So we need a 64-bit version of Windows 8.1 or higher in order to
install the distribution.  There are also 2 points to consider:
\begin{enumerate}
\item Choose a installation directory, and\label{item:1}
\item choose a \texttt{\acro{HOME}} directory\label{item:2}.
\end{enumerate}
Reg.\ item~\ref{item:1}, \msysii\ requires to extract the distribution into
a folder where the name consists of only \acro{ASCII} characters and no
spaces (also it makes good sense to put it in a path that doesn't use many
characters due to \texttt{\acro{PATH\_MAX}} being 260; \path|c:\msys64| is
ideal).  Reg.\ item~\ref{item:2}, I suggest to follow the advice
``\acro{ASCII} only, no spaces'' and possibly choose a directory other than
\path|c:\Users\<username>|.  I recommend to set the value of
\texttt{\acro{HOME}} environment variable to the directory chosen above
globally on Windows\Dash this is the only variable set outside \msysii.

Now we can fetch \msysii\ from \tburl{https://repo.msys2.org/distrib}.  We
want to install the portable version, so we download the
\path|msys2-x86_64-latest.tar.xz| archive and unpack it under \verb|c:\|.
In the file Explorer, we go to \path|c:\msys64| and double click on
\msysshell\ which opens a \acro{MSYS} shell, does the initial setup and
ideally shows:
\begin{flushleft}
  \includegraphics[%
    width=0.9\columnwidth,keepaspectratio
  ]{fig_msys-first-run}
\end{flushleft}

We follow the advice, close the window and double click \msysshell\ again.
U(Unix)---  auctex-article.tex    8%   L133  Git-master  (LaTeX/PS)
```

```
\section{Installing \texorpdfstring{\protect\msysii}{MSYS2} distribution}
\label{sec:inst-msys2-distr}

Before we start: As of March 2020, \msysii\ cannot be installed on a 32-bit
system.  And since January 2023, \msysii\ doesn't support Windows 7 and 8.0
anymore.  So we need a 64-bit version of Windows 8.1 or higher in order to
install the distribution.  There are also 2 points to consider:
\begin{enumerate}
\item Choose a installation directory, and\label{item:1}
\item choose a \texttt{\acro{HOME}} directory\label{item:2}.
\end{enumerate}
Reg.\ item~\ref{item:1}, \msysii\ requires to extract the distribution into
a folder where the name consists of only \acro{ASCII} characters and no
spaces (also it makes good sense to put it in a path that doesn't use many
characters due to \texttt{\acro{PATH\_MAX}} being 260; \path|c:\msys64| is
ideal).  Reg.\ item~\ref{item:2}, I suggest to follow the advice
``\acro{ASCII} only, no spaces'' and possibly choose a directory other than
\path|c:\Users\<username>|.  I recommend to set the value of
\texttt{\acro{HOME}} environment variable to the directory chosen above
globally on Windows\Dash this is the only variable set outside \msysii.

Now we can fetch \msysii\ from \tburl{https://repo.msys2.org/distrib}.  We
want to install the portable version, so we download the
\path|msys2-x86_64-latest.tar.xz| archive and unpack it under \verb|c:\|.
In the file Explorer, we go to \path|c:\msys64| and double click on
\msysshell\ which opens a \acro{MSYS} shell, does the initial setup and
ideally shows:
\begin{flushleft}
  \includegraphics[%
    width=0.9\columnwidth,keepaspectratio
  ]{fig_msys-first-run}
\end{flushleft}

We follow the advice, close the window and double click \msysshell\ again.
To update all packages we run the command \verb|pacman -Syu|.  We follow
the instructions and close the terminal if requested, then we start a new
terminal and update again with \verb|pacman -Syu|.  That's it!
TUGboat/auctex-article.tex  133:0  8%        LF UTF-8  LaTeX/PS  master  13:26 0.49
```

```
\section{Installing \texorpdfstring{\protect\msysii}{MSYS2} distribution}
\label{sec:inst-msys2-distr}

Before we start: As of March 2020, \msysii\ cannot be installed on a 32-bit
system.  And since January 2023, \msysii\ doesn't support Windows 7 and 8.0
anymore.  So we need a 64-bit version of Windows 8.1 or higher in order to
install the distribution.  There are also 2 points to consider:
\begin{enumerate}
\item Choose a installation directory, and\label{item:1}
\item choose a \texttt{\acro{HOME}} directory\label{item:2}.
\end{enumerate}
Reg.\ item~\ref{item:1}, \msysii\ requires to extract the distribution into
a folder where the name consists of only \acro{ASCII} characters and no
spaces (also it makes good sense to put it in a path that doesn't use many
characters due to \texttt{\acro{PATH\_MAX}} being 260; \path|c:\msys64| is
ideal).  Reg.\ item~\ref{item:2}, I suggest to follow the advice
``\acro{ASCII} only, no spaces'' and possibly choose a directory other than
\path|c:\Users\<username>|.  I recommend to set the value of
\texttt{\acro{HOME}} environment variable to the directory chosen above
globally on Windows\Dash this is the only variable set outside \msysii.

Now we can fetch \msysii\ from \tburl{https://repo.msys2.org/distrib}.  We
want to install the portable version, so we download the
\path|msys2-x86_64-latest.tar.xz| archive and unpack it under \verb|c:\|.
In the file Explorer, we go to \path|c:\msys64| and double click on
\msysshell\ which opens a \acro{MSYS} shell, does the initial setup and
ideally shows:
\begin{flushleft}
  \includegraphics[%
    width=0.9\columnwidth,keepaspectratio
  ]{fig_msys-first-run}
\end{flushleft}

We follow the advice, close the window and double click \msysshell\ again.
To update all packages we run the command \verb|pacman -Syu|.  We follow
the instructions and close the terminal if requested, then we start a new
terminal and update again with \verb|pacman -Syu|.  That's it!
```

```
TUGboat/auctex-article.tex  133:0  8%        LF UTF-8  LaTeX/PS  master  09:46 0.22
```

Figure 1: Emacs appearance: Vanilla Emacs (left); with doom-one theme, Windows dark mode and line numbers (middle); and doom-one-light theme (right), the latter two with disabled tool bar

5 Customizing Emacs

Emacs has the reputation for being highly customizable, and some even say Emacs users "customize to live". For basic usage and customization of Emacs, please refer to the Emacs manual [7], especially chapter 49. A good beginner's guide is also available.[4]

Next, we briefly mention some initialization code which is useful for installing and using AUCTEX. We will use the GNU Emacs Lisp Package Archive (ELPA) to install AUCTEX. The command `list-packages` gives for me a GPG error, but this can be circumvented by adding this to the Emacs init file:

```
Emacs init file
(setq package-check-signature nil)
```

We have to start server communicatoins for backward search in PDF files:

```
Emacs init file
(server-start)
```

We also like to select some text and then start typing where typed text replaces the selection, therefore:

```
Emacs init file
(delete-selection-mode 1)
```

Just in case we want to use a mouse to get a context menu, we add:

```
Emacs init file
(context-menu-mode 1)
```

Finally, if we want to change the font used by Emacs, we use the entry `Options` in the menu bar and go to `Set Default Font`. Figure 1 shows the result of some customization effort: On the left, we see Emacs showing this file without any adjustments, to the middle, a dark theme with Windows dark mode, and to the right, the way the author uses Emacs.

[4] `www.masteringemacs.org/article/beginners-guide-to-emacs`

The famous last words before entering the Emacs customizing realms:

- Try to use the Easy Customization Interface.
- Don't copy every snippet you find on the net into your init file.
- If you do that, read the manual and/or the docstring try to understand what the code does.
- If you don't understand the change, you probably don't need it.

6 Choosing a TEX mode for Emacs

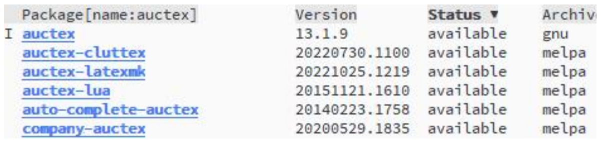
After installing Emacs, it's time to choose the appropriate support for authoring (LA)TEX files. Emacs has two major modes for this purpose: A built-in mode and the one provided by the AUCTEX package.[5]

So, which to choose? A general guideline might be: If you rely only on vanilla LATEX commands and environments, then try the built-in variant. If you will use large number of packages, want completion for the macros or environments and their (key-value) arguments, including syntax highlighting, and might define your own macros and environments and completion support for them is desired, then go for AUCTEX.

7 Installing AUCTEX

The modern and strongly recommended way of installing AUCTEX is by using the package manager integrated in Emacs to fetch it from ELPA. We type `M-x list-packages RET /n auctex RET`, put the cursor on `auctex`, press `i` and we see this:

[5] Each mode provides dedicated support for plain TEX, LATEX, DocTEX (for `.dtx` files) and SliTEX, but we will focus on LATEX.



```
  Package[name:auctex]        Version          Status ▾    Archiv
I auctex                      13.1.9           available   gnu
  auctex-cluttex              20220730.1100    available   melpa
  auctex-latexmk              20221025.1219    available   melpa
  auctex-lua                  20151121.1610    available   melpa
  auto-complete-auctex        20140223.1758    available   melpa
  company-auctex              20200529.1835    available   melpa
```

Now we hit `x` to execute the installation procedure. That's all. Using the ELPA version has several advantages. Besides being platform and OS independent, we will receive intermediate bugfix releases between major AUCTEX releases.

A word of caution: The way we installed AUCTEX, we must not have a line like this in our init file:

```
(load "auctex.el" nil t t)
```

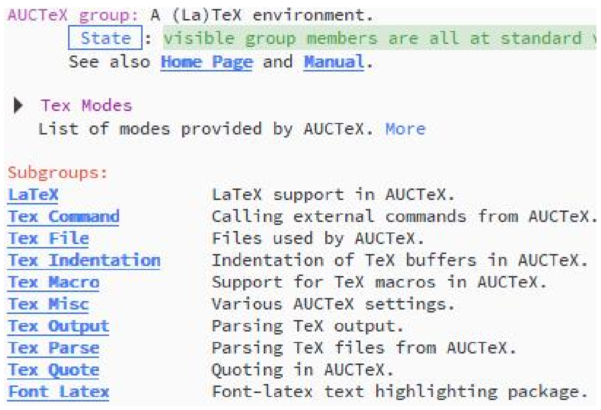
or even worse:

```
(require 'tex-site)
```

Having either such line in our init file may be harmful for the correct operations of AUCTEX.

8 Configuring AUCTEX

AUCTEX comes with a huge number of customization options; the figure below shows the various groups of options, some with subgroup(s).



```
AUCTeX group: A (La)TeX environment.
      State : visible group members are all at standard v
      See also Home Page and Manual.

▶ Tex Modes
   List of modes provided by AUCTeX. More

Subgroups:
LaTeX              LaTeX support in AUCTeX.
Tex Command        Calling external commands from AUCTeX.
Tex File           Files used by AUCTeX.
Tex Indentation    Indentation of TeX buffers in AUCTeX.
Tex Macro          Support for TeX macros in AUCTeX.
Tex Misc           Various AUCTeX settings.
Tex Output         Parsing TeX output.
Tex Parse          Parsing TeX files from AUCTeX.
Tex Quote          Quoting in AUCTeX.
Font Latex         Font-latex text highlighting package.
```

They are well described in the AUCTEX manual [10]. We will discuss some important options below which should be set before starting work.

Documents we edit can be a single file, or spread over many files consisting of a "master" file in which we include other files via LATEX macros like `\input` and `\include`. AUCTEX can deal with both single and multi-file projects and knows which file to compile via the variable `TeX-master`. This variable should be set in the Emacs init file and will also be inserted in each file's local variables. In general, it is a good idea to do:

Emacs init file
```
(setq-default TeX-master nil)
```

which means that when we create a new TEX file, AUCTEX will ask for the name of the "master" file associated with the buffer and insert a marker as a file variable in that file. For a single file project, it will look like this:

.tex file
```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
```

For a multi-file project, it might look like this:

.tex file
```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: "../phd-main"
%%% End:
```

Another important variable is `TeX-parse-self`. AUCTEX depends heavily on being able to extract information from the buffers by parsing them. Since parsing the buffer can be somewhat slow, the parsing is initially disabled. We enable it by adding the following line to our init file:

Emacs init file
```
(setq TeX-parse-self t)
```

This change means: Upon loading a ⟨*filename*⟩`.tex`, AUCTEX will look in an `auto` subdirectory for parsed information stored in ⟨*filename*⟩`.el`. If it finds that file, it is loaded and the information from it is applied to the current editing buffer. If there is no such file, AUCTEX parses the current buffer and applies that information to the buffer. The information applied consists of names of used packages, where AUCTEX loads its corresponding support files, user-defined macros and environments, defined labels for completion, etc. There is a catch here: AUCTEX doesn't distinguish among extensions of parsed files. So if we have a TEX file named, say, `geometry.tex`:

Example for geometry.tex
```
\documentclass{article}
\usepackage{xcolor}
\begin{document}
text
\end{document}
```

AUCTEX will save the information after parsing in `geometry.el`; upon the next loading of the saved `geometry.el`, it loads `article.el`, `xcolor.el` and the file `geometry.el` provided by AUCTEX itself which adds support for macros provided by `geometry.sty` — but we did not load that package. In general, we should always use distinct names for our TEX files in order to avoid this sort of clash.

A related option is `TeX-auto-save`. When set to non-`nil`, AUCTEX will parse the file and write the information each time the TEX file is saved. Again, this option is initially disabled. We can still force the parsing of the TEX file by pressing `C-c C-n` for `TeX-normal-mode`. This is often the best choice, as we will be able to decide when it is necessary to reparse the file.

If we use packages which define table environments and we want to put captions above the tables, we adjust the variable `LaTeX-top-caption-list`:

Emacs init file
```
(setq LaTeX-top-caption-list
      '("table"          "table*"
        "SCtable"        "SCtable*"
        "sidewaystable" "sidewaystable*"))
```

Finally, we tell AUCTeX to convert all tabs in multiple spaces, preserving the indentation, when we save a file:

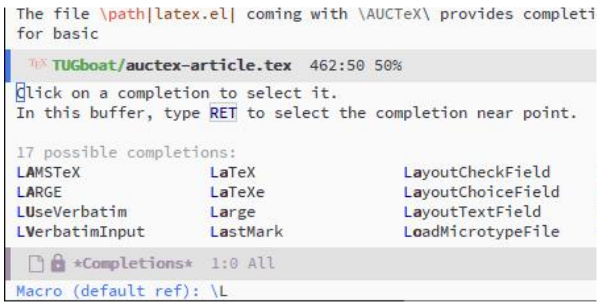
Emacs init file
```
(setq TeX-auto-untabify t)
```

9 Using AUCTeX

AUCTeX has an extensive manual which describes its usage in great detail [10]. Hence, we will discuss only some general usage aspects, focusing on completion of macros and environments with their arguments.

The file `latex.el` that comes with AUCTeX provides completion support for basic LaTeX macros and environments. As package files extend LaTeX's functionality, AUCTeX's style files extend its completion support. These style files are named after the package or class names used in a TeX file or the TeX file which was parsed, so (as mentioned above) `geometry.el` contains completion support for the macros provided by `geometry.sty`.

Completion support in AUCTeX is built around Emacs' *minibuffer completion*.[6] The entry points for inserting with completion are the functions `TeX-insert-macro` (bound to `C-c C-m` or `C-c RET`) and `LaTeX-insert-environment` (bound to `C-c C-e`). For example, this is what we see after hitting `C-c C-m L` followed by a `TAB` for completion candidates:

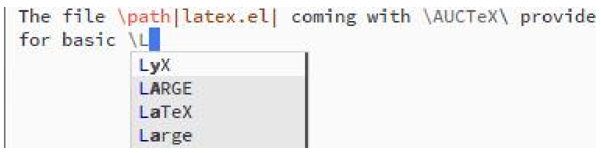


```
The file \path|latex.el| coming with \AUCTeX\ provides completi
for basic
 TUGboat/auctex-article.tex  462:50 50%
Click on a completion to select it.
In this buffer, type RET to select the completion near point.

17 possible completions:
LAMSTeX               LaTeX                 LayoutCheckField
LARGE                 LaTeXe                LayoutChoiceField
LUseVerbatim          Large                 LayoutTextField
LVerbatimInput        LastMark              LoadMicrotypeFile
 *Completions*  1:0 All
Macro (default ref): \L
```

AUCTeX presents the known candidates and we can narrow down the choices by typing further and hitting `RET` once we have the right macro which is inserted into the buffer and further arguments are queried, if applicable.

[6] gnu.org/software/emacs/manual/html_node/emacs/Completion.html

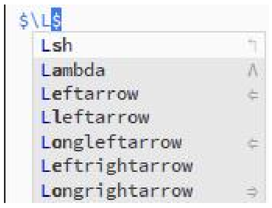
But sometimes we just want to insert the macro directly into the buffer, or find out we have forgotten a key-value pair in an argument where hitting the keystrokes described above will not help: we want *in-buffer completion*. As in the scenario above where we wanted to insert the `\LaTeX` macro, we can insert `\L` in the buffer followed by `TAB` and we get:



```
The file \path|latex.el| coming with \AUCTeX\ provide
for basic \L
          LyX
          LARGE
          LaTeX
          Large
```

where we can choose the macro and hit `RET` to insert.

AUCTeX also checks if we are in math mode and offers math symbols for completion. In order to get in-buffer completion, we need to install a package like `corfu`[7] or `company`[8] and configure it accordingly. It should be noted that in-buffer completion is not implemented in AUCTeX for all macro and environment arguments; this is work in progress.



```
$\L$
  Lsh              ↰
  Lambda           Λ
  Leftarrow        ⇐
  Lleftarrow
  Longleftarrow    ⇐
  Leftrightarrow
  Longrightarrow   ⇒
```

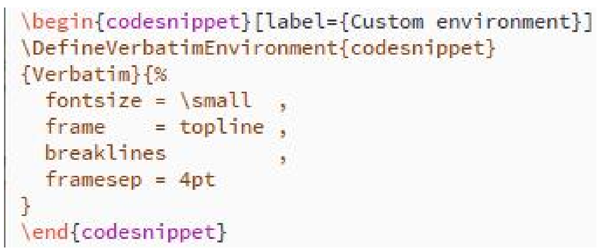
10 Hacking AUCTeX

One of AUCTeX's chief achievements is that its parser is "hackable", i.e., AUCTeX users and style files can extend the built-in parser with Lisp code. For example, this document uses the fvextra package, which loads fancyvrb in turn, and defines a custom verbatim environment, named `codesnippet`, like this:

Custom environment
```
\DefineVerbatimEnvironment{codesnippet}
{Verbatim}{%
  fontsize = \small   ,
  frame    = topline ,
  breaklines         ,
  framesep = 4pt
}
```

AUCTeX has a style file `fvextra.el`, which loads the style `fancyvrb.el` in turn, which contains code telling AUCTeX about the macro and its arguments defining a new verbatim environment. With the TeX code above in a file, AUCTeX sets its internal variables properly itself upon next parsing and no user intervention is needed. The new environment `codesnippet` is available when `C-c C-e` is hit, including completion and query for the optional key-value argument. Syntax highlighting support is also set automatically:

[7] github.com/minad/corfu
[8] company-mode.github.io



```
\begin{codesnippet}[label={Custom environment}]
\DefineVerbatimEnvironment{codesnippet}
{Verbatim}{%
  fontsize = \small  ,
  frame    = topline ,
  breaklines         ,
  framesep = 4pt
}
\end{codesnippet}
```

The general strategy for extending the parser is to write an AUCTEX style file where we:

- initialize the new entry to the parser by calling the `TeX-auto-add-type` lisp macro with its arguments;
- write a variable containing the regular expression which should be added to the parser and plug it into AUCTEX inside the hook;
- write a function which is run before parsing, resetting the results from the last parser run;
- write a function which is run after parsing, processing the results from the actual parser run.

We will discuss this process with two examples.

10.1 A simple example

The geometry package provides a facility to save the page dimensions as a ⟨*name*⟩ and load these dimensions later in the document. The macros are `\savegeometry` for saving the page dimensions, and `\loadgeometry` for loading. The AUCTEX style file `geometry.el` has the following code to parse the newly defined ⟨*name*⟩. First, a new entry for the parser is setup with:

geometry.el
```
(TeX-auto-add-type "geometry-savegeometry"
                   "LaTeX"
                   "geometry-savegeometries")
```

`TeX-auto-add-type` is a Lisp macro which takes two mandatory and one optional arguments: The first argument is a ⟨*name*⟩, which is prefixed by the second argument ⟨*prefix*⟩. Usually, ⟨*name*⟩ is composed as ⟨*package-macro*⟩ and ⟨*prefix*⟩ is the name of the engine or format used, in this case `LaTeX`. The third argument is the plural form of the first argument; by default just an `s` is added. The Lisp macro defines: the variable `LaTeX-auto-geometry-savegeometry` which holds the bare results after a successful parsing run; the function `LaTeX-geometry-savegeometry-list` which sorts and eliminates any dupes from `LaTeX-auto-geometry-savegeometry`; the variable `LaTeX-geometry-savegeometry-list` which holds the information returned by the function of the same name; and the function `LaTeX-add-geometry-savegeometries` which can be used to add new elements to `LaTeX-geometry-savegeometry-list`.

Next, `geometry.el` defines the variable `LaTeX-geometry-savegeometry-regexp`:

geometry.el
```
(defvar LaTeX-geometry-savegeometry-regexp
  '("\\\\savegeometry{\\([^}]+\\)}"
    1 LaTeX-auto-geometry-savegeometry))
```

which is a list of three elements: A string with the regular expression to match against, including a grouping construct for future reference, in this case the argument of `\savegeometry` with `{\\([^}]+\\)}`. The second element is an integer or a list of integers containing the number(s) of substring(s) matched, and finally the name of the variable to put the parsed substring(s) in. After this, a function is defined in preparation for parsing and is added to `TeX-auto-prepare-hook`:

geometry.el
```
(defun LaTeX-geometry-auto-prepare ()
  (setq LaTeX-auto-geometry-savegeometry nil))

(add-hook 'TeX-auto-prepare-hook
          #'LaTeX-geometry-auto-prepare t)
```

And finally, the defined regular expression is added to the parser with the function `TeX-auto-add-regexp` inside the hook. Also, two entries are defined for the LATEX macros:

geometry.el
```
(TeX-add-style-hook
 "geometry"
 (lambda ()
   (TeX-auto-add-regexp
    LaTeX-geometry-savegeometry-regexp)
   (TeX-add-symbols
    `("savegeometry"
      ,(lambda (optional)
         (let ((name (TeX-read-string
                      (TeX-argument-prompt
                       optional nil "Name"))))
           (LaTeX-add-geometry-savegeometries
            name)
           (TeX-argument-insert name
                                optional))))
    '("loadgeometry"
      (TeX-arg-completing-read
       (LaTeX-geometry-savegeometry-list)
       "Name")))))
```

The entry for `"savegeometry"` queries for a name and adds the user input to list of new names. The entry for `"loadgeometry"` retrieves all defined names and offers them as argument with completion.

10.2 A more complex example

For a more complex example, we look at the AUCTEX style file `enumitem.el` which contains code to parse new environments defined with the `\newlist` macro:

enumitem.el

```
(TeX-auto-add-type "enumitem-newlist" "LaTeX")

(defvar LaTeX-enumitem-newlist-regexp
  '("\\\\newlist{\\([^}]+\\)}{\\([^}]+\\)}"
    (1 2) LaTeX-auto-enumitem-newlist))
```

`\newlist` takes three arguments, but only the first two, a ⟨*name*⟩ and ⟨*type*⟩, are relevant. So the regular expression matches two arguments and both are added to the variable containing the results. Next, two functions are defined to prepare the parsing and process the results:

enumitem.el

```
(defun LaTeX-enumitem-auto-prepare ()
  (setq LaTeX-auto-enumitem-newlist nil))

(defun LaTeX-enumitem-auto-cleanup ()
  ;; \newlist{<name>}{<type>}{<depth>}
  ;; env=<name>, type=<type>
  (dolist (env-type
           (LaTeX-enumitem-newlist-list))
    (let* ((env  (car env-type))
           (type (cadr env-type)))
      (LaTeX-add-environments
       `(,env
         LaTeX-env-item-args
         [TeX-arg-key-val
          (LaTeX-enumitem-key-val-options)]))
      (when (member type '("description"
                           "description*"))
        (add-to-list
         'LaTeX-item-list
         `(,env . LaTeX-item-argument)))
      (TeX-ispell-skip-setcdr
       `((,env ispell-tex-arg-end 0))))))
```

The second function is the interesting one: Every user-defined environment is added to the list of known environments, including support for key-value query for the optional argument. For description-like environments, the optional argument of `\item` will be queried as well. And finally, the optional argument of the environment is ignored during spell-checking (see §14). These functions and the regular expression are added to AUCTEX with:

enumitem.el

```
(add-hook 'TeX-auto-prepare-hook
          #'LaTeX-enumitem-auto-prepare t)
(add-hook 'TeX-auto-cleanup-hook
          #'LaTeX-enumitem-auto-cleanup t)

(TeX-add-style-hook
 "enumitem"
 (lambda ()
   (TeX-auto-add-regexp
    LaTeX-enumitem-newlist-regexp)))
```

The techniques described above can also be used for any user-defined macros which define new macros and/or environments. The best approach is to put the LATEX macros inside a package and the corresponding Lisp code inside an AUCTEX style file saved in a directory which is part of `TeX-style-private`. This way, the Lisp code is loaded each time the custom package is requested with `\usepackage`.

11 Using preview-latex

preview-latex is a package embedding preview fragments into Emacs source buffers under the AUCTEX editing environment for LATEX. It uses preview.sty for the extraction of certain environments (most notably displayed formulas). preview-latex was originally written by David Kastrup and is now maintained by the AUCTEX team. It has an extensive manual describing the relevant aspects of usage and configuration [3].

12 Using RefTEX

RefTEX is a package for managing labels, references, citations and index entries for LATEX documents within Emacs. RefTEX has been bundled and preinstalled with Emacs since version 20.2. Originally written by Carsten Dominik, it is currently maintained by the AUCTEX team. RefTEX has an excellent manual describing its functionality and options [1].

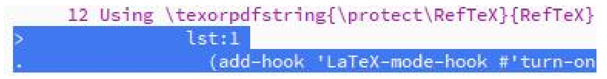
RefTEX can be used with both the built-in LATEX mode and AUCTEX. In order to plug RefTEX into AUCTEX, these two lines in our init file suffice:

Emacs init file

```
(add-hook 'LaTeX-mode-hook #'turn-on-reftex)
(setq reftex-plug-into-AUCTeX t)
```

The first line activates RefTEX automatically when AUCTEX is loaded and the second line turns on all RefTEX features within AUCTEX. The integration of the packages is seamless: AUCTEX checks in its style files if RefTEX is activated and updates RefTEX's variables with parsed elements where appropriate, and RefTEX's advanced mechanism for inserting labels and referencing them is used when AUCTEX's functions are invoked.

For example, within this document, a new environment `codesnippet` is defined (see §10). The fancyvrb package provides a key `reflabel` to define a new label to be used by `\pageref`. Now when we hit `C-c C-e code<TAB> RET ref<TAB> RET` without `=` and a value, AUCTEX completes the key and also adds `={lst:1}` to the key where the value is generated by RefTEX. We can now reference this label by hitting `C-c C-m RET pageref RET` and now AUCTEX delegates the request for labels to RefTEX and we choose the label type in the minibuffer with `l` and see the following:



```
      12 Using \texorpdfstring{\protect\RefTeX}{RefTeX}
>                lst:l
.                   (add-hook 'LaTeX-mode-hook #'turn-on
```

Similar things happen with citation macros.

Since the LaTeX release of October 2019, it is possible to use non-ASCII characters in labels such as `\label{eq:größer}`. With the standard setup, RefTeX will not allow us to enter such a label and complain about invalid characters. This behavior can be changed with the following addition to our Emacs init file:

```
Emacs init file
(setq reftex-label-illegal-re
      "[^-[:alnum:]_+=:;,.]")
```

13 Using a PDF viewer

On Windows, there are two TeX friendly PDF viewers: SumatraPDF[9] and Sioyek.[10] Both keep the PDF file unlocked, and both support SyncTeX. SumatraPDF has been around since 2006, Sioyek since 2021. We will use SumatraPDF. Installing SumatraPDF is easy: We fetch the portable version and unpack the single binary into `c:\msys64\usr\local\bin`. We run `mingw64.exe` and rename the file:

```
MinGW64 shell
$ cd /usr/local/bin
$ mv SumatraPDF-3.4.6-64.exe SumatraPDF.exe
```

Now we have to tell both parties, Emacs and SumatraPDF, about their counterparts. AUCTeX has built-in support for SumatraPDF, so there is not much to do but put this in our init file:

```
Emacs init file
(setq TeX-view-program-selection
      '((output-pdf "SumatraPDF")))
```

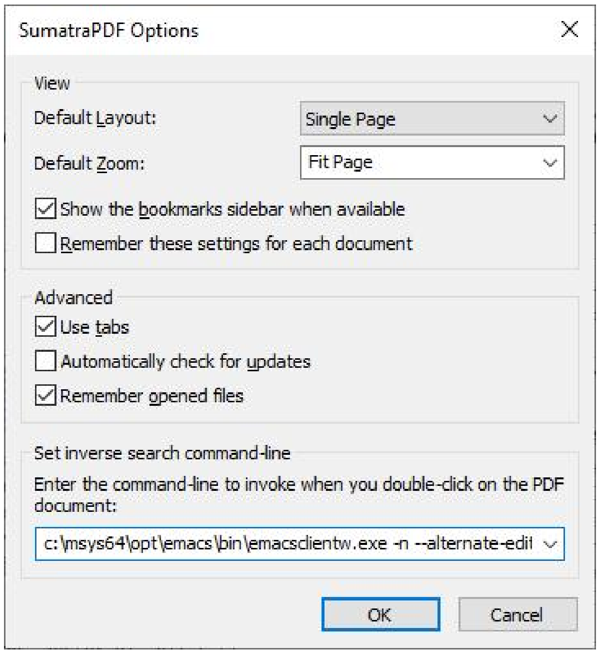
Emacs will find `SumatraPDF.exe` since it's installed in the MSYS2 file tree.

Next, under SumatraPDF options for inverse search command-line, we enter the following (except all on one line):

```
SumatraPDF options
c:\msys64\opt\emacs\bin\emacsclientw.exe -n
--alternate-editor=
c:\msys64\opt\emacs\bin\runemacs.exe
+%l "%f"
```

which means: Use the program `emacsclientw.exe` to connect to Emacs server, and if there is no Emacs server running, invoke `runemacs.exe` to open Emacs and connect to it. Note that this only works when SumatraPDF is invoked from a MinGW64 shell with:

9 `sumatrapdfreader.org`
10 `sioyek.info`



SumatraPDF Options

View

Default Layout: Single Page

Default Zoom: Fit Page

[✓] Show the bookmarks sidebar when available

[ ] Remember these settings for each document

Advanced

[✓] Use tabs

[ ] Automatically check for updates

[✓] Remember opened files

Set inverse search command-line

Enter the command-line to invoke when you double-click on the PDF document:

c:\msys64\opt\emacs\bin\emacsclientw.exe -n --alternate-edit

Figure 2: AUCTeX options for SumatraPDF, including inverse search.

```
MinGW64 shell
$ SumatraPDF.exe &
```

Or when invoked with `C-c C-v` from Emacs, everything works just fine.

Finally, we tell AUCTeX during editing to enable SyncTeX ("inverse search") when running the compiler; see figure 2. If we want to enable SyncTeX ad-hoc for a file, we can hit `C-c C-t C-s` which activates `TeX-source-correlate-mode` for the current file. If we want to have this mode activated for a specific file, we can add the following to the file:

```
.tex file
%%% Local Variables:
%%% mode: latex
%%% TeX-source-correlate-mode: t
%%% End:
```

And if we want to have the mode always enabled, we can customize the variable `TeX-source-correlate-mode` to `t`.

14 Using a spelling checker program

Emacs supports the external spell checkers Hunspell, Aspell, Ispell and Enchant. These programs are not part of Emacs and must be installed separately. We'll use Hunspell because it has the feature that we can use multiple language dictionaries at once. The complete setup consists of three parts:

- install the program itself;
- install the language dictionaries;
- set up Emacs to use the above.

Installing the program is easy: We run `msys2.exe` and enter:

MSYS shell
```
$ pacman -S mingw-w64-x86_64-hunspell
```

Next we need to create the directory where we will install the dictionaries, say under `/usr/local/share/hunspell`. We enter this in the shell and exit:

MSYS shell
```
mkdir -p /c/msys64/usr/local/share/hunspell
```

In our `~/.bashrc`, we add the following lines:

~/.bashrc
```
DICPATH=/c/msys64/usr/local/share/hunspell
WORDLIST=$HOME/.emacs.d/hunspell_default
export DICPATH WORDLIST
```

Next we download dictionaries for US English[11] and other languages.[12] We rename the `.oxt` extension to `.zip` so we can open the archive easily and we move the files with `.aff` and `.dic` extension into the `DICPATH` directory chosen above. Now we run `mingw64.exe` and enter:

MinGW64 shell
```
$ hunspell -D
```

Hunspell should report the available dictionaries in the `msys64` file tree.

Now we tell Emacs about Hunspell and add the following line to our init file:

Emacs init file
```
(setopt ispell-program-name "hunspell")
```

The next line tells Emacs about the default dictionary to use. E.g., for people preferring to write in German, it would be:

Emacs init file
```
(setq ispell-dictionary "deutsch8")
```

When we're writing LaTeX, we have to pass the `-t` option to Hunspell:

Emacs init file
```
(add-hook 'LaTeX-mode-hook
          (lambda ()
            (setq-local ispell-extra-args
                        '("-t"))))
```

We also set the name of our personal dictionary:

Emacs init file
```
(setq ispell-personal-dictionary
      (expand-file-name
       "~/.emacs.d/hunspell_default"))
```

This file must exist for Hunspell, but it can be an empty file. Finally, we define some key bindings to switch dictionaries:

[11] downloads.sourceforge.net/wordlist/hunspell-en_US-2020.12.07.zip

[12] extensions.libreoffice.org

Emacs init file
```
(keymap-global-set
 "C-c i e"
 (lambda ()
   (interactive)
   (ispell-change-dictionary "english")))

(keymap-global-set
 "C-c i d"
 (lambda ()
   (interactive)
   (ispell-change-dictionary "deutsch8")))

(keymap-global-set
 "C-c i a"
 (lambda ()
   (interactive)
   (require 'ispell)
   (ispell-set-spellchecker-params)
   (ispell-hunspell-add-multi-dic
    "de_DE,en_US")
   (ispell-change-dictionary
    "de_DE,en_US")))
```

Now we can invoke Hunspell inside Emacs with `M-x ispell` or inside AUCTeX with `C-c C-c Spell`. More information can be obtained from the Emacs manual.[13] AUCTeX provides a library `tex-ispell.el` which contains extensions for skipping certain macros, arguments and environments when spell checking. The supported packages are listed in the header of the library. These extensions are activated by default; they can be disabled by setting the value of `TeX-ispell-extend-skip-list` to `nil`.

15 Using Pygments

If we want to use the minted package, we have to install the additional software Pygments. We run `msys2.exe` and enter:

MSYS shell
```
$ pacman -S mingw-w64-x86_64-python-pygments
```

We can check the installation by running `mingw64.exe` and:

MinGW64 shell
```
$ which pygmentize.exe
```

which returns `/mingw64/bin/pygmentize.exe`.

minted requires that we pass the `-shell-escape` option to the LaTeX processor. This can be done by setting the AUCTeX variable `TeX-command-extra-options` as a file local variable:

.tex file
```
%%% Local Variables:
%%% mode: latex
%%% TeX-command-extra-options: "-shell-escape"
%%% End:
```

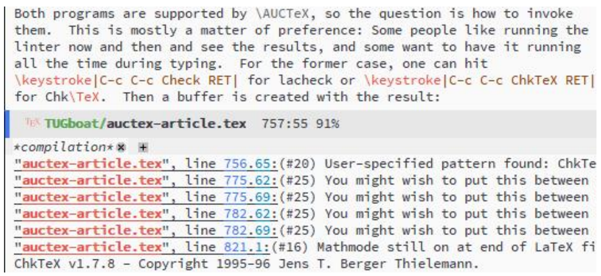
[13] gnu.org/software/emacs/manual/html_node/emacs/Spelling.html

AUCTEX has extensive support for the `minted` package, so using the package should work flawlessly.

16 Using a linter

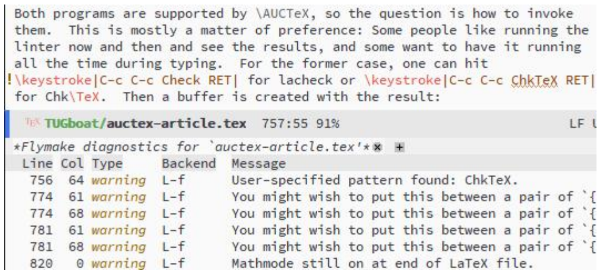
There are two linters available for LaTeX documents: lacheck[14] and ChkTEX.[15] Both of them are available with TEX Live as part of `collection-binextra`.

Both programs are supported by AUCTEX, so the question is how to invoke them. This is mostly a matter of preference: Some people like running the linter now and then and see the results, and some want to have it running all the time during typing. For the former case, one can hit `C-c C-c Check RET` for lacheck or `C-c C-c ChkTeX RET` for ChkTEX. Then a buffer is created with the result:



```
Both programs are supported by \AUCTeX, so the question is how to invoke
them.  This is mostly a matter of preference: Some people like running the
linter now and then and see the results, and some want to have it running
all the time during typing.  For the former case, one can hit
\keystroke|C-c C-c Check RET| for lacheck or \keystroke|C-c C-c ChkTeX RET|
for Chk\TeX.  Then a buffer is created with the result:

TUGboat/auctex-article.tex  757:55 91%
*compilation*
"auctex-article.tex", line 756.65:(#20) User-specified pattern found: ChkTe
"auctex-article.tex", line 775.62:(#25) You might wish to put this between
"auctex-article.tex", line 775.69:(#25) You might wish to put this between
"auctex-article.tex", line 782.62:(#25) You might wish to put this between
"auctex-article.tex", line 782.69:(#25) You might wish to put this between
"auctex-article.tex", line 821.1:(#16) Mathmode still on at end of LaTeX fi
ChkTeX v1.7.8 - Copyright 1995-96 Jens T. Berger Thielemann.
```

For the latter case of on-the-fly syntax checking, Emacs provides a minor mode called Flymake which is supported by AUCTEX. It can be activated with `M-x flymake-mode`. The same result now looks like this:



```
 Both programs are supported by \AUCTeX, so the question is how to invoke
 them.  This is mostly a matter of preference: Some people like running the
 linter now and then and see the results, and some want to have it running
 all the time during typing.  For the former case, one can hit
!\keystroke|C-c C-c Check RET| for lacheck or \keystroke|C-c C-c ChkTeX RET|
 for Chk\TeX.  Then a buffer is created with the result:

 TUGboat/auctex-article.tex  757:55 91%                                  LF
*Flymake diagnostics for `auctex-article.tex'*
 Line Col Type     Backend  Message
  756  64 warning  L-f      User-specified pattern found: ChkTeX.
  774  61 warning  L-f      You might wish to put this between a pair of `{
  774  68 warning  L-f      You might wish to put this between a pair of `{
  781  61 warning  L-f      You might wish to put this between a pair of `{
  781  68 warning  L-f      You might wish to put this between a pair of `{
  820   0 warning  L-f      Mathmode still on at end of LaTeX file.
```

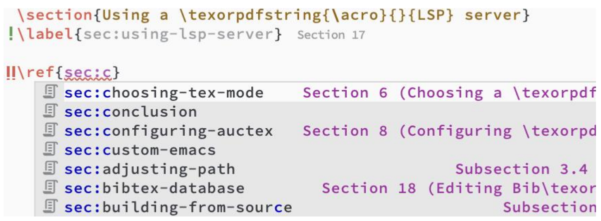
Note also the visual effects we get with Flymake. Flymake has also an extensive manual [8].

17 Using a LSP server

Emacs 29 ships with a new library called `eglot.el` (for *E*macs Poly*glot*) which is a built-in client for the *Language Server Protocol* (LSP). LSP is a standardized communications protocol between source code editors and language servers — programs external to Emacs which analyze the source code on behalf of Emacs. We can now open a source file and type `M-x eglot`, presuming that an appropriate language server is installed. Eglot comes with a manual describing the details [9].

Currently, two LSP servers are available for LaTeX: TexLab[16] and Digestif.[17] Installing TexLab is easy: We download the correct version from project's page and unpack `texlab.exe` into `c:\msys64\usr\local\bin`. Digestif is part of TEX Live and distributed as `digestif.exe`.

`eglot` knows about both TexLab and Digestif, so we can activate a LSP server by hitting `M-x eglot` and choosing the one we want in case both servers are installed. That's it. The next figure shows an example for this document with TexLab which adds the section number to the `\label` macro and provides annotated completion for the `\ref` macro.



```
 \section{Using a \texorpdfstring{\acro}{}{LSP} server}
!\label{sec:using-lsp-server}  Section 17

!!\ref{sec:c}
    sec:choosing-tex-mode    Section 6 (Choosing a \texorpdf
    sec:conclusion
    sec:configuring-auctex   Section 8 (Configuring \texorpd
    sec:custom-emacs
    sec:adjusting-path                      Subsection 3.4
    sec:bibtex-database       Section 18 (Editing Bib\texor
    sec:building-from-source                     Subsection
```

18 Editing BIBTEX databases

Emacs has a built-in major mode for editing BIBTEX files which is used when we open a `.bib` file. This major mode supports both BIBTEX and BIBLaTEX; BIBTEX is the default. This can be changed by customizing the variable `bibtex-dialect`:

Emacs init file
```
(setopt bibtex-dialect 'biblatex)
```

Once the mode is active, it is easy to use the menus or the context menu to add new entries and operate on the fields.

19 Miscellaneous settings

This section describes various other settings which should make the daily work easier.

AUCTEX provides in-buffer completion which can be activated with the `TAB` key. The `TAB` key is somewhat overloaded since it is also used for indentation. The operation of `TAB` can be controlled with the variable `tab-always-indent`. We can set this in our init file:

Emacs init file
```
(setq tab-always-indent 'complete)
```

which means: `TAB` first tries to indent the current line, and if the line was already indented, then try to complete the thing at point.

TEX Live provides a batch script `tlmgr.bat` for managing the distribution. Being a batch file, it is not possible to run the script inside a MinGW64 shell. We can change this by putting this small

[14] ctan.org/pkg/lacheck
[15] ctan.org/pkg/chktex

[16] github.com/latex-lsp/texlab
[17] github.com/astoff/digestif

snippet under `c:\msys64\usr\local\bin` and name it `tlmgr`:

tlmgr script

```
#!/bin/sh
# This is a small wrapper around tlmgr.bat
# Note the double // for escaping /
cmd.exe //c tlmgr.bat "$@"; exit $?
```

When we're inside the MinGW64 shell, hitting `TAB` provides completion for executables and/or file names. Under Windows, also files with `.dll` suffix are offered for executable completion. We change this with this line in our `~/.bashrc`:

~/.bashrc

```
export EXECIGNORE=*.dll
```

`EXECIGNORE` is a colon-separated list of glob patterns to ignore when completing on executables. This is an MSYS2[18] feature.

Another handy idea is to alias `emacsclient` to run `emacsclient.exe` with some options:

~/.bashrc

```
alias emacsclient='emacsclient -n \
--alternate-editor=runemacs'
```

20 Conclusion

TeX has been around for some time now, and so has Emacs. Both carry the original ideas of their developers, but they have also managed to evolve over the decades. Emacs can be set up to look modern,[19] but more importantly, it also supports modern techniques to support users to write LaTeX documents.

With the advent of MSYS2, it is easily possible to build Emacs from the source on Windows, so an initial barrier to getting the program is gone. With AUCTeX, a configurable major mode for LaTeX is available which can be installed easily as a package from ELPA. RefTeX is a great tool for managing labels and citations and is bundled with Emacs. Other tools around the editor such as spell-checker, PDF viewer, Pygments, linter, etc., can be integrated into the editing environment without trouble.

One new feature in Emacs 29 is the built-in client for LSP servers which works out of the box for available language servers. The support for this feature is expected to grow. Another new feature in Emacs 29 is the built-in support for the incremental parsing library Tree-sitter. The usage of Tree-sitter with Emacs for TeX editing is an area which needs more exploration in the future.

Overall, Emacs provides a very good environment for editing (LA)TeX documents using up-to-date tools and techniques which can be easily set up on Windows.

Acknowledgments

I'm grateful to the AUCTeX development team and Óscar Fuentes (MSYS2 contributor) for their comments on this article.

References

[1] C. Dominik. *RefTeX — Support for LaTeX labels, references, citations and index entries with GNU Emacs.* `gnu.org/software/auctex/manual/reftex.index.html`

[2] Emacs. Building and Installing Emacs on 64-bit MS-Windows using MSYS and MinGW-w64. `git.savannah.gnu.org/cgit/emacs.git/tree/nt/INSTALL.W64`

[3] D. Kastrup, J.Å. Larsson, et al. *preview-latex — A LaTeX preview mode for AUCTeX in Emacs.* `gnu.org/software/auctex/manual/preview-latex.index.html`

[4] S. Monnier, M. Sperber. Evolution of Emacs Lisp. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020. `doi.org/10.1145/3386324`

[5] MSYS. MSYS Software Distribution and Building Platform for Windows. `msys2.org`

[6] R. Stallman. Why Open Source Misses the Point of Free Software. `gnu.org/philosophy/open-source-misses-the-point.en.html`, 2007–2021.

[7] R. Stallman, et al. *GNU Emacs Manual (updated for Emacs version 29.1)*, 1985–2023. `gnu.org/software/emacs/manual/emacs`

[8] J. Távora, P. Kobiakov. *GNU Flymake.* `gnu.org/software/emacs/manual/flymake.html`

[9] J. Távora, E. Zaretskii. *Eglot: The Emacs Client for the Language Server Protocol.* `joaotavora.github.io/eglot/`

[10] K.K. Thorup, P. Abrahamsen, et al. *AUCTeX: A sophisticated TeX environment for Emacs.* `gnu.org/software/auctex/manual/auctex.index.html`

◇ Arash Esbati
Germany
`arash (at) gnu dot org`

[18] Cygwin, to be more precise.

[19] Depending on the definition, which currently seems to be Microsoft Visual Studio Code.

Wikipedia to LaTeX, PDF, EPUB and ODT

Dirk Hünniger

Abstract

The MediaWiki2Latex program converts wiki content to LaTeX and other formats. It has been more than 10 years since we last mentioned it in *TUGboat*,[1] so there is quite a bit of news to report.

1 Introduction

A wiki is a great way of working on a document with a distributed group of authors. MediaWiki is the Wiki system used by Wikipedia, owned by the Wikimedia Foundation.

Wikipedia used to provide ways to download the contents in various formats. Their idea was to finance the service by selling printed copies of articles as books; this did not prove cost effective, due to a lack of demand. In turn the download functions were not maintained, and eventually removed. Many efforts were undertaken to re-enable exporting, but all such development stopped years ago.

Our open source approach, as a hobby project with no financing, recently celebrated its tenth anniversary, and is still under active development. It is deployed on a server kindly provided to us by the Wikimedia Foundation.

2 User experience and web service

We dropped the development of a specific binary package for Windows. Instead we offer a docker file that can be run on any operating system. In addition, we still update and support the package for the Debian Linux distribution, which is included in many other Linux distributions too. We furthermore provide an online conversion service at `mediawiki2latex.wmflabs.org`.

This service takes the url to a wiki article and outputs a resulting file for download. We also added additional output file formats. You can choose between a PDF file compiled with LaTeX, its respective LaTeX source code as a zip file, as well as the word processing format ODT and the ebook format EPUB.

Some Wiki pages extensively use HTML tricks to create browser-viewable graphics, such as election diagrams or maps with marked positions. For those cases we optionally offer rendering the tables via the Chromium engine. It is also possible to process collections of wiki articles to a single output file.

[1] *TUGboat* 34:2, "Converting Wikipedia articles to LaTeX", `tug.org/TUGboat/tb34-2/tb107huenniger.pdf`

3 Command line interface

We offer a command line interface that can be installed locally as a docker container. This provides the same features as the web service described above.

There is a feature in MediaWiki called "templates" which is similar to `\newcommand` in LaTeX. In the command line interface you can specify parsing of the wiki source code, instead of the HTML generated by MediaWiki. Here you can provide a mapping of templates to LaTeX commands which allows you to customize the output in various ways. Also you can let MediaWiki expand the templates to Wiki syntax and use it as input for MediaWiki2LaTeX. When wiki syntax is processed, MediaWiki2LaTeX will resolve references inside the document to sections and page numbers.

4 Technical details

MediaWiki2LaTeX is written entirely in the purely functional programming language Haskell. The image processing is done by ImageMagick in C++. Recently we added http2 multiplexing and compression using curl for the download of images and their respective contributor information, which resulted in a speedup of a factor of two to five, with the highest speedups on articles containing many small images. We also attempted to implement http multiplexing with multi-threading, but this did not work due to servers denying multiple connections.

Furthermore, many smaller bugs have been fixed; they are tracked in a detailed change log in the source package, so we will not discuss them here. The runtime and memory usage of various parts of the program were improved, also tracked in the change log. The documentation has also been improved.

Due to the remaining problem of no free font covering the whole Unicode range, we implemented an algorithm to switch between various fonts during the XeLaTeX run as needed. This allowed MediaWiki2LaTeX to become an official part of the Debian Linux distribution, which was not possible with the computationally combined font which we used before. A man page and Makefile were added to support easy packaging of the software. Support for http has been dropped in favor of https. Finally, a progress bar was added in the web interface as well as a graphical user interface for Debian.

◇ Dirk Hünniger
Emil-Schweitzer Straße S 10
D-47506 Neukirchen-Vluyn, Germany
`dirk.hunniger (at) googlemail dot com`
`https://de.wikibooks.org/wiki/Benutzer:Dirk_Huenniger`

Fast regression testing of TeX packages: Multiprocessing and batching

Vít Starý Novotný, Marei Peischl

Abstract

In the version 3.0.0 of the Markdown package for TeX, the number of regression tests increased from 143 to 783. This caused the tests to run for up to 15 hours, which slowed down our development cycle. In this article, we describe a novel technique for batching test files that reduced our testing time from 15 hours to just 15 minutes. With batching, the amount of time that is spent on actual testing increased from 5% to 97%. When combined with multiprocessing on 32 CPUs, our batching technique achieved a speed increase of up to 161 times compared to running without any multiprocessing or batching.

1 Introduction

Small TeX packages, typically developed in a single iteration rather than through ongoing updates, can depend on user feedback to maintain the code. However, this approach has its limitations. Larger projects, especially those that are continuously developed, require a more robust solution. Automated regression tests are crucial in these cases. They ensure that any changes, either in the code itself or its external dependencies, do not alter the expected behavior of the code.

The Markdown package for TeX also features a set of regression tests. These tests, designed to be completed in just a few minutes, provide immediate feedback and are automatically conducted on any updates submitted to the package's GitHub repository.

After the implementation of the CommonMark standard in version 3.0.0 of the Markdown package, the number of tests increased from 143 to 783 (about a 5.5-fold increase). This caused the tests to take up to 15 hours to run, using free GitHub-hosted runners, too slow to provide any benefit to developers.

In order to increase the testing speed, we implemented a novel technique for batching test files and we added self-hosted runners with up to 12 CPUs. After these changes, the tests finish in about 15 minutes, which is a 60-fold speed increase and which makes the tests practically useful to developers.

In this article, we describe the testing framework of the Markdown package. In sections 2 through 4, we describe the definition files, techniques, and strategies used in our framework. In Section 5, we describe the details of our implementation. In sections 6 and 7, we describe our experiments and their results. In Section 8, we discuss prior work related to our framework. We conclude in Section 9 by summarizing our contributions and outlining future work.

2 Definition files

In the future, an AI agent might examine the code of a TeX package and identify any incorrect behavior. For the moment, regression testing requires the manual creation of many definition files that describe the expected behavior and how it should be validated.

In this section, we describe the definition files used in our framework: test files, formats, commands, and templates.

2.1 Test files

The Markdown package converts markdown text to TeX commands. To validate the conversion, our framework redefines these TeX commands to produce output in the `.log` file, which we then examine.

A *test file* consists of a) TeX code that configures the Markdown package, b) markdown text, and c) the expected output in the `.log` file.

As an example, `strike-through.test` tests the strike-through syntax extension:

```
\markdownSetup
  { strikeThrough }   a
<<<
Hello ~~world~~!   b
>>>
BEGIN document
strikeThrough: world   c
END document
```

strike-through.test

2.2 Formats, commands, and templates

The Markdown package supports several combinations of TeX formats and engines. For each TeX format, there are also several ways to input markdown text. Our framework ensures that a markdown text always produces the same output.

A *format* consists of one or more a) *commands* that can be used to typeset documents in a TeX format using different TeX engines and b) *templates* that specify the different ways in which markdown text can be input with the TeX format.

An example format `plain` contains commands for the pdfTeX, XeTeX, and LuaTeX engines:

```
pdftex --shell-escape
↪  TEST_FILENAME
xetex --shell-escape
↪  TEST_FILENAME
luatex TEST_FILENAME[1]
```

COMMANDS.m4

[1] The Markdown package uses Lua to parse markdown text. Whereas LuaTeX can execute Lua code directly, other TeX engines must use the shell of the operating system to

The format `plain` also contains two templates. One uses the `\markdownInput` TeX macro and the other one uses the `\markdownBegin` and `End` TeX macros:

```
\input markdown
\input TEST_SETUP_FILENAME
\markdownInput
↪ {TEST_INPUT_FILENAME}
\bye
```

`input.tex.m4`

```
\input markdown
\input TEST_SETUP_FILENAME
\markdownBegin
undivert(TEST_INPUT_FILENAME)
\markdownEnd
\bye
```

`verbatim.tex.m4`

2.3 Materialized templates and commands

During testing, the texts `TEST_SETUP_FILENAME` and `TEST_INPUT_FILENAME` in templates are replaced with names of auxiliary files that contain the TeX code and the markdown text parts of a test file, respectively. Also, the text `undivert(TEST_INPUT_FILENAME)` is replaced with the literal markdown text from the test file. After the replacement, we say that the template has been *materialized*.

Here is the template `verbatim.tex.m4` from Section 2.2 after it has been materialized with the test file `strike-through.test` from Section 2.1:

`verbatim.tex.m4` + `strike-through.test` ⤵

```
\input markdown
\input test-setup.tex
\markdownBegin
Hello ~~world~~!
\markdownEnd
\bye
```

`verbatim.tex`

```
\markdownSetup
  { strikeThrough }
```

`test-setup.tex`

After a template has been materialized, the text `TEST_FILENAME` in all commands is replaced with the filename of the materialized template. After the replacement, the command has also been materialized.

execute Lua code. Since accessing the shell is a security risk, users must express their consent by writing `--shell-escape`.

Here are the commands `COMMANDS.m4` from Section 2.2 after they have been materialized:

`COMMANDS.m4` + `verbatim.tex` ⤵

```
pdftex --shell-escape
↪ verbatim.tex
xetex --shell-escape
↪ verbatim.tex
luatex verbatim.tex
```

`COMMANDS`

During testing, the materialized commands are executed. Each command produces a `.log` file, which is compared to the expected output from the test file.

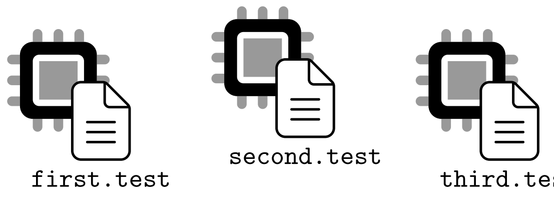
3 Computational techniques

While testing all combinations of test files, templates, and commands ensures comprehensive coverage of all potential configurations, it can be time-consuming.

In this section, we describe the computational techniques of multiprocessing, the batching of test files, and how they increase the speed of testing in our framework. Furthermore, the batching of test files raises challenges with load balancing and the attribution of errors. We discuss the challenges and describe the techniques of batch size limiting and batch splitting to address them.

3.1 Multiprocessing

Whereas TeX uses only a single CPU, modern PCs can contain several CPUs. Therefore, we can increase the speed of testing by using *multiprocessing*, where each CPU processes a different test file:



Using N CPUs increases testing speed up to N times.

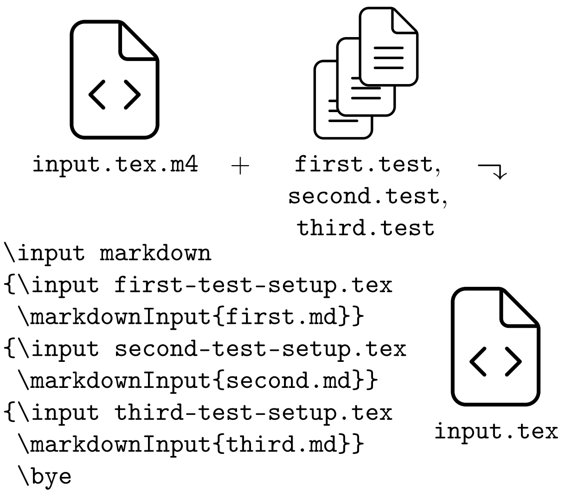
3.2 Batching of test files

At the beginning of a document, TeX initializes packages, fonts, Lua scripts, and other assets, which slows down testing:

```
\input markdown          } slow
\input test-setup.tex    ⎫
\markdownBegin           ⎬ fast
Hello ~~world~~!         ⎪
\markdownEnd             ⎭
\bye
```

`verbatim.tex`

To increase the speed of testing, we can amortize the cost of initialization by materializing a template with a *batch* of several test files:

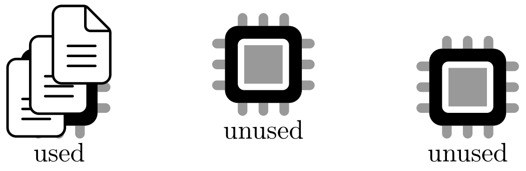


```
\input markdown
{\input first-test-setup.tex
 \markdownInput{first.md}}
{\input second-test-setup.tex
 \markdownInput{second.md}}
{\input third-test-setup.tex
 \markdownInput{third.md}}
 \bye
```

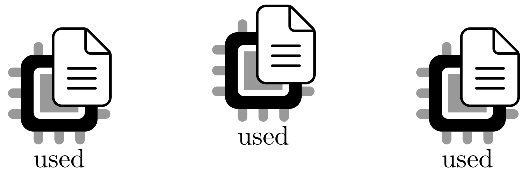
Batching N test files decreases initialization cost N times. How much this speeds up testing depends on the ratio between the time spent on initialization and the time spent on processing the rest of the template.

3.3 Batch size limiting

When we use both multiprocessing and batching with large batch sizes, most CPUs will be unused:



We can *limit the batch size* so that all CPUs are used:

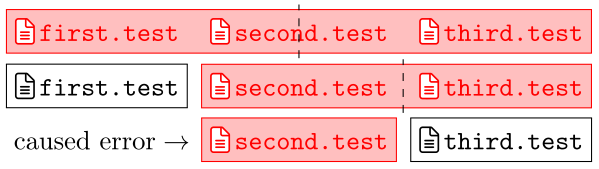


Limiting the batch size increases the speed of testing, because every used CPU has less work to do.[2]

3.4 Batch splitting

When we test batches of test files, a `.log` file is split into sections corresponding to individual test files and compared with the expected test file outputs. However, if a fatal error occurs, the `.log` file may become malformed. To find the test file responsible for the error, we repeatedly *split the batch* using binary search.

Here is how we would split a batch of test files `first.test`, `second.test`, and `third.test`, where `second.test` causes a fatal error:



First, we try processing all files together but we encounter a fatal error. Therefore, we divide the files into two groups: one with `first.test` and the other with `second.test` and `third.test`. Processing these separately, we again face a fatal error in the group with `second.test` and `third.test`. We then split this group into two individual files, `second.test` and `third.test`, and we process them. The fatal error occurs with `second.test`, which we identify as the cause of the error.

When only one test file out of N files in a batch causes a fatal error, batch splitting executes at most $2(\log_2 N + 1)$ commands. This is less than or equal to N for sufficiently large batch sizes $N \geq 8$. Therefore, in the presence of no more than a few fatal errors, batching is still faster than sequential processing.

4 Error handling strategies

Developers and maintainers have different needs when it comes to the handling of errors. Whereas developers need immediate feedback during the development of new features, maintainers require a comprehensive summary of all errors when they deal with unexpected breakage.

In this section, we describe the error handling strategies of developer- and maintainer-oriented testing and updating test files. We also discuss how these strategies increase the speed of development and decrease maintenance costs.

[2] However, the CPUs do more work overall, because every used CPU has to pay the initialization cost and more CPUs are used. Therefore, limiting the batch size increases the speed of testing but decreases energy efficiency.

4.1 Developer-oriented testing

In the practice of test-driven development, before adding a new feature, developers first write new test files that describe the expected behavior of the feature. Then, they develop the feature until all test files have passed. At the beginning, old test files will pass, whereas new test files will fail. At the end, all test files, both old and new, should pass.

In order to increase the speed of development, tests should fail fast to provide immediate feedback. Therefore, developers can configure our framework to start with the new test files, which are the most likely to fail, and stop at the first error rather than wait until all tests have finished.

For example, imagine a TeX package with two test files: `first.test` and `second.test`. For simplicity, the package has only one format with one template and with three commands for the pdfTeX, XeTeX, and LuaTeX engines. Before the development of a new feature, developers add a new test file `third.test` and they run the tests with the following results:

	pdfTeX	XeTeX	LuaTeX
`first.test`			
`second.test`			
`third.test`	✗		

Since `third.test` was new, it was tested first and immediately failed, providing immediate feedback to developers. This is how we test all updates submitted to the GitHub repository of the Markdown package.

4.2 Maintainer-oriented testing

Tests can fail not just during the development of new features but also during maintenance. These errors are often caused by changes to external dependencies such as TeX engines, formats, and packages.

In order to decrease maintenance costs, tests should provide comprehensive feedback. Therefore, maintainers can configure our framework to always process all test files and produce a helpful summary of all errors.

Continuing the example from the previous section, developers finish the new feature and release an updated version of their package on CTAN. However, after a month, the tests fail with the following results:

	pdfTeX	XeTeX	LuaTeX
`first.test`	✓	✗	✓
`second.test`	✓	✗	✓
`third.test`	✓	✗	✓

At a glance, the summary shows that the errors are related to the XeTeX engine. This is how we test the Markdown package every week.

4.3 Updating test files

Although test-driven development is well-suited to adding new features, fundamental changes to the code may require that existing test files are updated as well. Furthermore, writing test files before the development of a feature can be difficult, especially for complex features with incomplete requirements.

In order to increase the speed of development, developers can configure our framework to *update test files* instead of failing. Developers can make fundamental changes and our framework will update the expected outputs in test files to match the actual output. Furthermore, developers can also develop a feature, write partial test files for the feature that contain only the TeX code and markdown text, and use our framework to fill in the expected output. Then, developers can review the changes and determine whether they are correct.

Our framework will update a test file only if all templates and commands produce consistent outputs. In the example from the previous section, the command for the XeTeX engine failed for all test files, whereas the other commands did not. Therefore, our framework would not update any test files and fail.

5 Implementation

Before Markdown 3.0.0, our framework was implemented by a Bash script `test.sh`; see Listing 1.

At first, `test.sh` processed test files sequentially and did not use the computational techniques from Section 3 to increase the speed of testing. Since Markdown 2.4.0, we used the GNU Parallel command-line tool [7] to implement multiprocessing:

```
$ find -name '*.test' | parallel ./test.sh
```

Out of the error handling strategies from Section 4, `test.sh` could only update test files.

While Bash is convenient for simple programs, more complicated programs are better written in a more expressive language. In Markdown 3.0.0, we rewrote our framework from Bash to Python 3 [4].

Out of the techniques and strategies from Sections 3 and 4, the higher expressiveness of Python allowed us to implement the batching of test files, developer- and maintainer-oriented testing, and batch splitting. Furthermore, Python's built-in support for multiprocessing allowed us to stop using GNU Parallel and implement batch size limiting.

test.sh

```
#!/bin/bash
set -o errexit -o pipefail -o nounset
BUILDDIR="$(mktemp -d)"
trap 'rm -rf "$BUILDDIR"' INT TERM
for TESTFILE; do
  printf 'Testfile %s\n' "$TESTFILE"
  for FORMAT in templates/*/; do
    printf '  Format %s\n' "$FORMAT"
    for TEMPLATE in "${FORMAT}"*.tex.m4; do
      printf '    Template %s\n' "$TEMPLATE"
      m4 -DTEST_FILENAME=test.tex <"$FORMAT"/COMMANDS.m4 |
      (while read -r COMMAND; do
        printf '      Command %s\n' "$COMMAND"

        # Set up the testing directory.
        cp support/* "$TESTFILE" "$BUILDDIR"
        cd "$BUILDDIR"
        sed -r '/^\s*<<<\s*$/{x;q}' \
          <"${TESTFILE##*/}" >test-setup.tex
        sed -rn '/^\s*<<<\s*$/,/^\s*>>>\s*$/{/^\s*(<<<|>>>)\s*$/!p}' \
          <"${TESTFILE##*/}" >test-input.md
        sed -n '/^\s*>>>\s*$/,${/^\s*>>>\s*$/!p}' \   a
          <"${TESTFILE##*/}" >test-expected.log
        m4 -DTEST_SETUP_FILENAME=test-setup.tex \
           -DTEST_INPUT_FILENAME=test-input.md <"$OLDPWD"/"$TEMPLATE" >test.tex

        # Run the test, filter the output and concatenate adjacent lines.
        eval "$COMMAND" >/dev/null 2>&1 ||
          printf '        Command terminated with exit code %d.\n' $?    b
        touch test.log
        sed -nr '/^\s*TEST INPUT BEGIN\s*$/,/^\s*TEST INPUT END\s*$/{
          /^\s*TEST INPUT (BEGIN|END)\s*$/!H
          /^\s*TEST INPUT END\s*$/{s/.*//;x;s/\n//g;p}
        }' <test.log >test-actual.log

        # Compare the expected outcome against the actual outcome.
        diff -a -c test-expected.log test-actual.log ||              c
        # Uncomment the below lines to update the testfile.
#          (sed -n '1,/^\s*>>>\s*$/p' <"${TESTFILE##*/}" &&
#           cat test-actual.log) >"$OLDPWD"/"$TESTFILE" ||     d
           false

        # Clean up the testing directory.
        cd "$OLDPWD"
        find "$BUILDDIR" -mindepth 1 -exec rm -rf {} +
      done)
    done
  done
done
rm -rf "$BUILDDIR"
```

Listing 1: The shell script `test.sh` that implemented the testing framework of the Markdown package before version 3.0.0. For each test file, `test.sh` a) materializes templates in a temporary directory, b) executes materialized commands, c) compares the `.log` file against the expected output, and d) optionally updates the test file.

6 Experiments

In this section, we describe our experiments with multiprocessing and batching. In our experiments, we aimed to answer the following research questions:

1. What is the speed benefit of multiprocessing?
2. What is the speed benefit of batching test files?
3. What is the speed benefit of batch size limiting?

To answer the questions, we tested the Markdown package with different CPU counts and batch sizes:

- Numbers of CPUs: 1, 2, 4, 8, 16, and 32
- Batch sizes: 1, 2, 4, 8, …, 256, 512, and 1024

To ensure reliability of our findings, we repeated each test configuration five times and we measured the median testing time to control for sample variance. Our experimental code is available online. [6]

We tested the Markdown package at Git commit `7613632` from August 21, 2023. At this commit, the Markdown package contained 783 test files. For each test file, 14 commands were materialized and executed: four for plain TEX, four for LATEX, and six for ConTEXt MkIV. Therefore, TEX formats were initialized up to $14 \cdot 783 = 10{,}962$ times during testing.

To show the speed benefit of batch size limiting, we deactivated it in our experiments, thereby highlighting the speed reduction caused by its absence.

We ran the experiments for 33 days on a shared GNU/Linux server with 400 GB of RAM and 80 CPUs, each at 2.1 GHz.

7 Results

Figure 1 shows the results of our experiments. In this section, we discuss the results and how they relate to the three research questions outlined in the previous section.

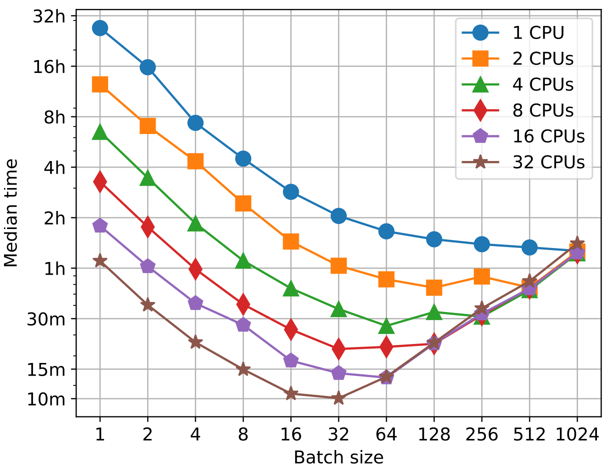


Figure 1: The median testing times for different numbers of CPUs and batch sizes

7.1 Multiprocessing

With batch size 1, the testing speed scales almost linearly with the number of CPUs, as we would expect: Whereas with 1 CPU, the median testing time is 27 hours and 2 minutes, it is only 1 hour and 6 minutes with 32 CPUs (about 24-fold speed-up).[3]

7.2 Batching of test files

With 1 CPU, the testing speed also scales almost linearly with the batch size, up to a point. Whereas with batch size 1, the median testing time is 27 hours and 2 minutes, it is only 4 hours and 30 minutes with batch size 8 (about 6-fold speed-up), and 1 hour and 20 minutes with batch size 512 (about 21-fold speed-up). This indicates that initialization dominates the testing time.

To better understand the relationship between the initialization and the testing time, we can solve the following series of equations:

$$14 \cdot (783 \cdot (X + Y)) = 27 \text{ hours and } 2 \text{ minutes}$$
$$14 \cdot (X + 783 \cdot Y) = 1 \text{ hour and } 16 \text{ minutes}$$

On the left-hand side of the equations, the variable X stands for the mean time that it takes to initialize a TEX format and the variable Y stands for the mean time that it takes to process the markdown text from a single test file. On the right-hand side of the equations are the median testing times with 1 CPU and batch sizes 1 (above) and 1024 (below).

The solution shows that whereas it takes a full $X \approx 8.47$ seconds to initialize a TEX format, it takes only $Y \approx 0.41$ seconds to process a markdown text. In other words, without batching, 95% of time is spent on initialization and only 5% on actual testing; with batching, up to 97% of time is spent on testing.

7.3 Batch size limiting

The speed improvements from multiprocessing and batching are additive, up to a point. Whereas with 1 CPU and batch size 1, the median testing time is 27 hours and 2 minutes, it is only 10 minutes with 32 CPUs and batch size 32 (about 161-fold speed-up).

When the number of CPUs multiplied by the batch size exceeds the number of test files (783), we cannot use all CPUs and the testing speed decreases. Whereas with 32 CPUs and batch size 16, the median testing time is only 11 minutes, it is 1 hour and 24 minutes with the same number of CPUs and batch size 1024 (about 8-fold slow-down). Our framework prevents this effect by limiting the batch size.

[3] The reason that we did not achieve the theoretical 32-fold speedup is likely tasks from other users on our server.

8 Related work

Besides our framework, there exist other frameworks for regression testing of TEX packages. Furthermore, the computational techniques in our framework are often adapted from previous work in other fields.

In this section, we discuss the regression testing framework of the l3build package management system and we compare it with our framework. Furthermore, we discuss the origin of the batching of test files and batch splitting.

8.1 The l3build package

The l3build package [2, 9, 8, 1] provides a comprehensive system for TEX package management that also includes a regression testing framework.

Whereas our framework is written in Python, l3build is written in Lua. Each language presents its own set of strengths and weaknesses. On one hand, every modern installation of TEX includes a Lua interpreter, which makes l3build more accessible for TEX users compared to our framework. On the other hand, unlike Python, Lua has no built-in support for multiprocessing. Therefore, l3build users must use external tools like GNU Parallel to use multiple CPUs for testing, whereas our framework can use multiple CPUs out of the box.

In our framework, each test file is designed to hold a single self-contained test that avoids modifying the global state. This design allows for straightforward grouping of tests into batches, where each test is isolated from others using TEX groups, as we discussed in Section 3.2.

In l3build, a single test file may contain multiple tests. These tests can be interdependent, creating a challenge in separating them from their files. Additionally, these tests might change the global state, which poses a risk of unexpected conflicts when tests are grouped into batches. Due to these complexities, l3build does not support the batching of tests. Nonetheless, the practice of including multiple tests in a single test file can be seen as a form of manual batching that amortizes the cost of initialization.

Both our framework and l3build support the updating of test files [1, Section 2.7]. This allows developers to automatically generate parts of test files when they make fundamental changes to the code or when they develop complex new features, as discussed in Section 4.3.

8.2 Batching and batch splitting

The techniques of batching and batch splitting were perhaps first used with TEX in the ARQMath competitions in large-scale indexing of math formulae.

In the first ARQMath competition, the MIRMU team used the LATEXML tool to convert TEX formulae to XML [5, Section 2.2]. Due to speed issues when processing each formula separately, MIRMU processed them in batches. However, a single error would cause the loss of an entire batch. Therefore, MIRMU used batch splitting to recover correct formulae after an error [3].

In the third ARQMath competition, the organizers used the same techniques to provide math formulae in the XML format to all participants.

9 Conclusion

Larger TEX packages commonly use regression tests to ensure code integrity over time. In this study, we explored techniques for speeding up the regression testing of TEX packages. We have shown that batching test files can improve the testing efficiency from 5% to 97%. We have also shown that multiprocessing on 32 CPUs combined with the batching of test files increases testing speed up to 161 times.

The lessons learned from our work are as follows:

- Whereas TEX uses only a single CPU, modern PCs can contain several CPUs. Multiprocessing can increase testing speed by using these CPUs.
- Writing small isolated test files is convenient for authors but carries a high initialization cost during testing. For TEX packages such as the Markdown package, where tests are easy to isolate, the batching of test files can be used to recover the initialization cost.
- Whereas developers need tests to fail as fast as possible, maintainers benefit from a comprehensive summary of all errors.

We hope that our practical lessons will improve the regression testing practices used in the development and maintenance of TEX packages. With fast regression testing, developers can quickly introduce new features, while maintainers can proactively address emerging issues before they affect users.

Disclaimer

No wolves were harmed in the making of this article.

Acknowledgements

We would like to gratefully acknowledge the Natural Language Processing Center at the Faculty of Informatics, Masaryk University, Brno, who kindly provided computational resources for our experiments.

We also wish to extend our special thanks to Joseph Wright, Ben Frank, Barbara Beeton, and Karl Berry for their invaluable insights and thorough review of our work. Their expertise and thoughtful feedback have been instrumental in shaping the final manuscript.

Donation request

Despite our breakthroughs in testing speed, additional computational resources would greatly accelerate the development and maintenance of the Markdown package for TeX.

We graciously invite donations of GitHub self-hosted runners, particularly those hosted on GNU/Linux servers with at least 16 GB of RAM and 12 CPUs. For more information, please contact us by email. Donors will be acknowledged in the project documentation, with the option to remain anonymous upon request.

References

[1] LaTeX project team. l3build: A testing and building system for (La)TeX, Nov. 2023. `ctan.org/pkg/l3build`

[2] F. Mittelbach, W. Robertson, LaTeX3 team. l3build: A modern Lua test suite for TeX programming. *TUGboat* 35(3):287–293, 2014. `tug.org/TUGboat/tb35-3/tb111mitt-l3build.pdf`

[3] V. Novotný. ARQMath data preprocessing, June 2020. `github.com/MIR-MU/ARQMath-data-preprocessing/blob/main/scripts/latex_tsv_to_cmml_and_pmml_tsv.py`

[4] V. Novotný. Implement batching and summarization to unit tests, Jan. 2023. `github.com/witiko/markdown/issues/245`

[5] V. Novotný, P. Sojka, et al. Three is better than one. In *CEUR Workshop Proceedings: ARQMath task at CLEF conference*, vol. 2696, pp. 1–30, Thessaloniki, Greece, 2020. CEUR-WS. `ceur-ws.org/Vol-2696/paper_235.pdf`

[6] V. Starý Novotný. Measure the speed of tests with different numbers of processes and batch sizes, Oct. 2023. `github.com/Witiko/markdown/blob/main/experiments/2023-10-12-test-batching`

[7] O. Tange. GNU Parallel: The command-line power tool. *USENIX Mag*, 36(1):42–47, 2011. Available from `doi.org/10.5281/zenodo.8278274`.

[8] J. Wright. l3build: The beginner's guide. *TUGboat* 43(1):40–43, 2022. `tug.org/TUGboat/tb43-1/tb133wright-l3build.pdf`

[9] J. Wright, LaTeX3 team. Automating LaTeX(3) testing. *TUGboat* 36(3):234–236, 2015. `tug.org/TUGboat/tb36-3/tb114wright.pdf`

◇ Vít Starý Novotný
Studená 453/15
Brno 63800, Czech Republic
`witiko (at) mail dot muni dot cz`
`github.com/witiko`

◇ Marei Peischl
Gneisenaustr. 18
Hamburg 20253, Germany
`marei (at) peitex dot de`
`peitex.de`

Illustrating finite automata with Grail+ and Ti*k*Z

Alastair May, Taylor J. Smith

Abstract

In this article, we discuss a new software tool that interacts with Grail+, a library of automata-theoretic command-line utilities. Our software, the Grail+ Visualizer, takes the textual representation of a finite automaton produced by Grail+ and generates Ti*k*Z code to illustrate the finite automaton, with automatic layout of states and transitions. In addition to giving an overview of the basics of automata theory and Grail+, we discuss how the Grail+ Visualizer works in detail and suggest avenues for future work.

1 Introduction

Grail+ is a C++ library of command-line utilities that performs symbolic manipulation of various models of finite automata, regular expressions, and finite languages. Each utility, called a *filter* in Grail+ terminology, can either handle input directly or be piped together to create a chain of filters. Grail+ consists of nearly one hundred filters that can compute common operations or procedures that a theoretical computer scientist might want to perform; among many other tasks, these filters can enumerate the elements of a language (`fmenum`), convert a finite automaton to an equivalent regular expression (`fmtore`) and vice versa (`retofm`), and minimize (`fmmin`) or determinize (`fmdeterm`) a finite automaton. These predefined filters, together with the ability to chain filters together, allow users to perform thousands of formal language and automata-theoretic tasks.

The original environment, Grail, was developed by Darrell Raymond at the University of Waterloo and Derick Wood at the University of Western Ontario [6]. Following major changes instituted in version 3.0, the name of the environment changed to Grail+ and coordination of the development work was taken over by Sheng Yu, again at the University of Western Ontario. Presently, Grail+ is being developed and maintained by Cezar Câmpeanu at the University of Prince Edward Island, and the current stable version is 3.4.5 [2].

2 Finite automata in Grail+

Since Grail+ runs in a command-line interface, all input and output is plain text. For models that can be represented naturally in text, like regular expressions, Grail+ follows the conventional notation in theoretical computer science; for instance, the union of regular expressions `a` and `b` is `a + b`, and the concatenation of these regular expressions is `ab`.

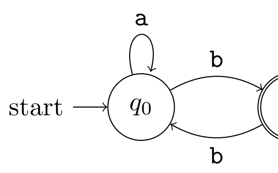


Figure 1: An example of a finite automaton.

For non-textual models like finite automata, however, Grail+ defines its own convention.

As an example, consider the finite automaton depicted in Figure 1. This finite automaton takes a *word* (or *string*) consisting of `a`s and `b`s as its input. It has two *states*, labelled q_0 and q_1, and *transitions* on these states labelled by `a` and `b`. The contents of the word determine which transition the finite automaton follows at any given step of its computation; for example, if the finite automaton reads the symbol `b` in the word while it is in state q_0, then it will transition to state q_1 before reading the next symbol. The state q_0 is the *initial state*, or the state where the computation of the finite automaton begins. The state q_1 is a *final* or *accepting state*; if, after reading all symbols in the word, the finite automaton finds itself in state q_1, then it accepts that word. The set of all words accepted by a finite automaton is the *language* of that finite automaton.

In Grail+, finite automata are represented as lists of *instructions*. Each individual instruction consists of three pieces of information: a source state, a label, and a sink state. Additionally, there are two special *pseudo-instructions* to indicate which states are initial and which states are final. Returning to Figure 1, this finite automaton would be represented in Grail+ as the following list:

```
(START) |- 0
0 a 0
0 b 1
1 a 1
1 b 0
1 -| (FINAL)
```

There is no particular ordering to the instructions in the lists produced by Grail+ as output, and no ordering is enforced when giving a list to Grail+ as input. Additionally, while the example given here has one initial state and one final state, a finite automaton may contain multiple initial and/or final states.

3 Typesetting finite automata

For authors who wish to include illustrations of finite automata in their documents, the most straightforward way to do so — apart from importing an external image file — is to use Ti*k*Z and the PGF

doi.org/10.47397/tb/45-1/tb139may-automata

```
\begin{tikzpicture}[node distance=2cm]
    \node[state, initial] (q0) {$q_{0}$};
    \node[state, accepting, right of=q0] (q1) {$q_{1}$};
    \path[->] (q0) edge[loop above] node[above] {\texttt{a}} (q0);
    \path[->] (q0) edge[bend left] node[above] {\texttt{b}} (q1);
    \path[->] (q1) edge[bend left] node[below] {\texttt{b}} (q0);
    \path[->] (q1) edge[loop above] node[above] {\texttt{a}} (q1);
\end{tikzpicture}
```

Figure 2: The (human-written) Ti*k*Z code producing the finite automaton in Figure 1.

package [9] to create the illustration. Ti*k*Z includes an automata drawing library with special shapes and styles specific to finite automata (see Chapter 43 of the Ti*k*Z & PGF manual [8]; see also the article in *TUGboat* 44:1 by Igor Borja [1]). An example of Ti*k*Z code using the automata drawing library is given in Figure 2.

Creating illustrations from scratch using Ti*k*Z puts the decision-making entirely in the user's hands, allowing for ample customization in layout, style, and other aspects. At the same time, creating illustrations from scratch using Ti*k*Z puts the decision-making *entirely* in the user's hands, leading to potential pitfalls. For particularly large or complicated finite automata, laying out states and transitions in an aesthetically pleasing way can become very difficult. Thus, using external software to assist in constructing and laying out the finite automaton can make the illustration process easier.

Andrew Mertz, William Slough, and Nancy Van Cleave wrote in *TUGboat* 35:2 about methods of illustrating computer science concepts using LaTeX packages [5]. In particular, in Section 7 of their article, the authors discuss typesetting automata using JFLAP [7], which is a Java software package for manipulating finite automata and formal languages. In contrast to Grail+, JFLAP uses a graphical user interface, and it is capable of exporting illustrations of finite automata to various image formats. The `jflap2tikz` package [4] additionally allows users to convert a finite automaton produced by JFLAP to Ti*k*Z code that can be included in a LaTeX document.

4 Motivation

While JFLAP is a popular software package, and while it is capable of handling more theoretical models of computation than Grail+ currently handles — namely grammars, pushdown automata, and Turing machines — there are reasons why users may still prefer to work with Grail+. For one, Grail+ can be run on any computer that is capable of compiling C++ code, and the suite can be customized and extended by anyone who is capable of writing C++ code. Subjectively, users may find the text-based command-line interface of Grail+ to be faster or easier to use than a point-and-click graphical user interface. Lastly, while software like JFLAP emphasizes pedagogy and learning about formal languages and automata theory, researchers and practitioners may value Grail+ for its efficient implementation and wider array of features specific to finite automata. (Together with these reasons, Canadians may uniquely value Grail+ for being made-in-Canada software.)

What Grail+ lacks, however, is a way to render its textual output in a more human-friendly form. It can be extremely difficult, especially with finite automata having many states or many transitions, for a user to parse the textual output and to gain an understanding of the structure of the finite automaton without representing it visually. Given that Ti*k*Z has a built-in automata drawing library, and that both the output from Grail+ and the Ti*k*Z code to produce an illustration of a finite automaton follow a fixed, pre-specified format, Ti*k*Z is a natural candidate for automatically laying out illustrations of finite automata produced by Grail+.

5 The Grail+ Visualizer

Our software tool, the Grail+ Visualizer [3], offers a more human-friendly — and beginner-friendly — way to construct, manipulate, and display automata. Working alongside Grail+, our visualizer transforms the textual output of Grail+ into Ti*k*Z code that can be either typeset and displayed on its own or inserted into an existing LaTeX document.

The visualizer software is written in Bash and can be run directly from a command-line interface, just like Grail+ itself. The software can therefore act as the final link in a chain of Grail+ filters, providing an immediate visual indication of the result.

5.1 How the visualizer works

The Grail+ Visualizer takes as input the textual representation of a finite automaton produced by Grail+ (i.e., the list of instructions) and parses the text to extract state labels, which are stored in a list. At the same time, state type labels are stored in an auxiliary list to distinguish whether a particular state is initial, final, or both. Transitions are also parsed and stored in three lists: one containing source state labels, one containing transition labels, and one containing sink state labels.

The visualizer then begins to lay out states and transitions. Each state is placed at coordinate (x, y) on a square grid according to the following procedure:

1. Assign x-coordinates to states in the order they are read from the input; that is, the first state read from input is assigned $x = 0$, the second state is assigned $x = 1$, and so on.
2. Initialize an all-zero array A of size $|Q|$, where $|Q|$ is the number of unique states identified in the input processing step. Indices of A correspond to x-coordinates of states; for example, $A[0]$ corresponds to the state having x-coordinate 0. Entries in A correspond to y-coordinates of states. At this point, each state has an initial y-coordinate of 0.
3. For each pair of states p and q connected by a transition in the finite automaton, where p has x-coordinate i, q has x-coordinate j, and assuming $i < j$ without loss of generality:
   (a) Denote by m the maximum entry in the subarray $A[i..j]$.
   (b) Increment m by 1.
   (c) Compare the value m to the entries $A[i]$ (i.e., the y-coordinate of state p) and $A[j]$ (i.e., the y-coordinate of state q). The largest of these three values will be the new y-coordinate of both states p and q.

These coordinates are used to generate a list of Ti*k*Z `\node`s, which is written to the output file. The label of each node corresponds to the state label stored during the input processing step. If a state is distinguished as initial or final, then the appropriate option (`initial` or `accepting`, respectively) is added to the corresponding node.

Next, where a transition exists from a source state p to a sink state q, the visualizer writes a Ti*k*Z `\path` to the output file producing a directed edge from p to q. This path is labelled by the transition label stored during the earlier input processing step. Some special cases are also handled during this step:

- Where there exists a transition from a state p to itself, the visualizer adds the option `loop above` to the corresponding path.
- Where there exist multiple transitions between a source state p and a sink state q, those multiple transitions are consolidated into a single path. The labels of the multiple transitions are obtained via pattern-matching existing lines of the output file and stored in a temporary variable. The existing paths are deleted, and a new path labelled by the stored transition labels is written to the output file.

At this stage, the Ti*k*Z code is complete, and the output file is ready for compilation. The file is typeset by PDFLaTeX and can be opened by a PDF viewer immediately, or the Ti*k*Z code may be copied into another LaTeX document.

5.2 An example

To demonstrate some of the capabilities of the Grail+ Visualizer, consider the following list of instructions produced by Grail+:

```
(START) |- 0
(START) |- 6
(START) |- 3
0 a 1
1 c 3
1 d 5
6 b 7
7 c 9
7 d 11
3 -| (FINAL)
5 -| (FINAL)
9 -| (FINAL)
11 -| (FINAL)
```

In this list of instructions, we have eight states unordered and numbered non-consecutively (e.g., there is a state `1` and a state `3`, but no state `2`). Of these states, three are initial states and four are final states, and state `3` is both initial and final. There is also a total of six transitions.

The Grail+ Visualizer begins by parsing the list of instructions and extracting the state labels from these instructions. From this input, the visualizer produces two preliminary lists: one of state labels and one of state types. At this stage, the state labels list may contain duplicates. For instance, state `3` appears three times in the preliminary state label list, corresponding to its appearances in lines 3, 5, and 10 of the input list of instructions.

The visualizer also extracts transition data and produces an additional three lists:

```
 0  1  1  6  7  7
 a  c  d  b  c  d
 1  3  5  7  9  11
```

From top to bottom, these three lists indicate the transition source states, the transition labels, and the transition sink states.

Duplicate state labels are then removed from the preliminary lists created earlier. In this example, the unique state labels are identified in the order `0`, `6`, `3`, `1`, `5`, `7`, `9`, and `11`, and the lists produced by the Grail+ Visualizer are as follows:

`0`	`6`	`3`	`1`	`5`	`7`	`9`	`11`
S	S	B	–	F	–	F	–

The top list contains state labels, while the bottom list contains state types. States may have one of four types: "S" denotes an initial state, "F" denotes a final state, "B" denotes a state that is both initial and final, and "–" denotes an undistinguished state.

Next, state positions are calculated according to the procedure outlined in Section 5.1. This procedure assigns the following y-coordinates to states:

`0`	`6`	`3`	`1`	`5`	`7`	`9`	`11`
1	4	0	3	0	6	0	6

From top to bottom, these lists indicate the state label and that state's y-coordinate. Recall that the state's x-coordinate is its index in the list, so state `0` is at position $(0, 1)$, state `6` is at position $(1, 4)$, and so on.

At this point, the various lists created by the visualizer are then used to produce the Ti*k*Z code corresponding to the automaton. The final Ti*k*Z code produced by the visualizer is shown in Figure 3, and typesetting the code produces the automaton shown in Figure 4.

6 Conclusions and future work

In this article, we introduced the Grail+ Visualizer, explained how the visualizer interacts with Grail+ to produce a typeset illustration of a finite automaton, and worked through an example demonstrating how the visualizer works. We hope that this software tool stimulates a greater interest in both automata theory and the Grail+ library.

There remain some areas for improvement in the visualizer software. For instance, the layout procedure could be optimized to place states sharing many transitions closer to one another or to avoid large numbers of crossing transitions. For certain finite automata, a more optimized layout procedure might use a "system of springs" technique inspired by that of Tutte [10]. Another desirable feature might allow the user to specify some degree of customization for the output; say, in setting the colours of states, the minimum distance between states, or the exact positions of certain states. Lastly, implementing the Grail+ Visualizer as a LaTeX package rather than as a separate piece of software would greatly simplify the workflow for users. This could allow a user to write a command like `\grailautomaton{instruction_list}` that would generate and include Ti*k*Z code within any LaTeX document at typesetting time.

Acknowledgements

The work done by the first author of this article was supported by an Alley Heaps undergraduate research internship provided by St. Francis Xavier University.

References

[1] I. Borja. An introduction to automata design with Ti*k*Z's automata library. *TUGboat* 44(1):102–107, 2023. `tug.org/TUGboat/tb44-1/tb136prado-automata.pdf`

[2] Department of Computer Science, University of Prince Edward Island. Theory of Computing Software Server. `grail.smcs.upei.ca`.

[3] A. May, T.J. Smith. Grail+ Visualizer. `github.com/flarelabstfx/Grail-Visualisation`.

[4] A. Mertz, W. Slough. The `jflap2tikz` package. `ctan.org/pkg/jflap2tikz`.

[5] A. Mertz, W. Slough, N. Van Cleave. Typesetting figures for computer science. *TUGboat* 35(2):179–191, 2014. `tug.org/TUGboat/tb35-2/tb110mertz.pdf`

[6] D. Raymond, D. Wood. Grail: A C++ library for automata and expressions. *Journal of Symbolic Computation*, 17(4):341–350, 1994.

[7] S.H. Rodger, T.W. Finley. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones & Bartlett Publishers, Sudbury, MA, 2006.

[8] T. Tantau, et al. *The TikZ and PGF Packages: Manual for Version 3.1.10*, January 2023. `mirrors.ctan.org/graphics/pgf/base/doc/pgfmanual.pdf`

[9] The PGF/Ti*k*Z Team. The `pgf` package. `ctan.org/pkg/pgf`.

[10] W.T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, s3-13:743–767, 1963.

◇ Alastair May
Department of Computer Science
St. Francis Xavier University
Antigonish, NS, B2G 2W5 Canada
`x2016owd (at) stfx dot ca`

◇ Taylor J. Smith
Department of Computer Science
St. Francis Xavier University
Antigonish, NS, B2G 2W5 Canada
`tjsmith (at) stfx dot ca`
`https://people.stfx.ca/tjsmith/`
ORCID 0000-0001-7838-3409

```
\begin{tikzpicture}[node distance=2cm]

\node[state,initial] (0) at (0,1) {$0$};
\node[state,initial] (6) at (1,4) {$6$};
\node[state,initial,accepting] (3) at (2,0) {$3$};
\node[state] (1) at (3,3) {$1$};
\node[state,accepting] (5) at (4,0) {$5$};
\node[state] (7) at (5,6) {$7$};
\node[state,accepting] (9) at (6,0) {$9$};
\node[state] (11) at (7,6) {$11$};
\path[->] (0) edge[] node[align=center, anchor=center, above, sloped] {a} (1);
\path[->] (1) edge[] node[align=center, anchor=center, above, sloped] {c} (3);
\path[->] (1) edge[] node[align=center, anchor=center, above, sloped] {d} (5);
\path[->] (6) edge[] node[align=center, anchor=center, above, sloped] {b} (7);
\path[->] (7) edge[] node[align=center, anchor=center, above, sloped] {c} (9);
\path[->] (7) edge[] node[align=center, anchor=center, above, sloped] {d} (11);

\end{tikzpicture}
```

Figure 3: Ti*k*Z code produced by the Grail+ Visualizer from the example input of Section 5.2.

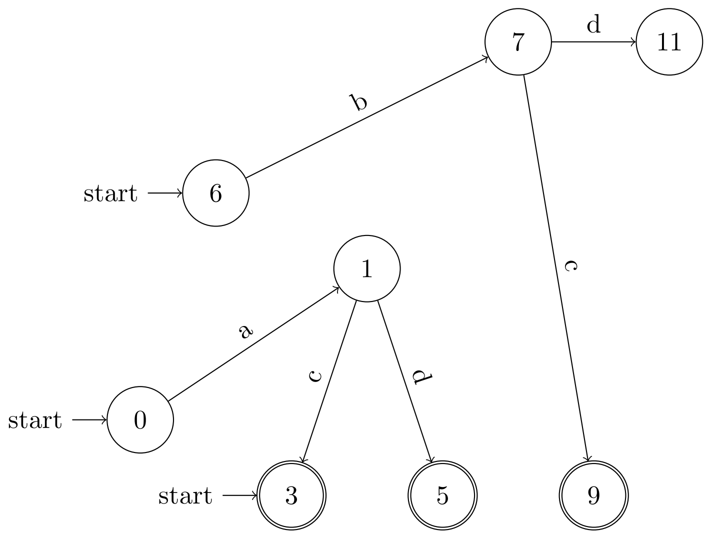


Figure 4: The illustration produced by compiling the Ti*k*Z code in Figure 3.

Including PDF files

Hans Hagen

Rendering glyphs in a PDF happens based on information in the page stream. In such a stream we find font triggers that use identifiers like `F1` and afterwards placement operators inject shapes by referring to an index in the font, say hexadecimal `0003`, and that index can refer to any shape. The identifier is resolved via the `Font` entry in the `Resources` dictionary of the page:

```
5 0 obj
<<
    /Contents 3 0 R
    /Resources << /Font << /F1 1 0 R >>
                  /ProcSet 2 0 R >>
    ...
    /Type /Page
>>
endobj
```

Following the `F1` reference we end up at:

```
1 0 obj
<<
    /BaseFont /TNTUFI+LMRoman10-Regular
    /DescendantFonts [ 11 0 R ]
    /Encoding /Identity-H
    /Subtype /Type0
    /ToUnicode 14 0 R
    /Type /Font
>>
endobj
```

and when we then descend into the first entry in the `DescendantFonts` array we come to:

```
11 0 obj
<<
    /BaseFont /TNTUFI+LMRoman10-Regular
    /CIDSystemInfo << /Ordering (Identity)
                      /Registry (Adobe)
                      /Supplement 0 >>
    /FontDescriptor 9 0 R
    /LMTXRegistry 8 0 R
    /Subtype /CIDFontType0
    /Type /Font
    /W 10 0 R
>>
endobj
```

The `W` entry value, rather short relative to the other keys, refers to an array object that holds the widths of the glyphs so that the viewer knows how much to advance; the `FontDescriptor` points to the shapes; and `LMTXRegistry` will be discussed later. These examples come from typesetting:

```
\starttext
    \startTEXpage
        test
    \stopTEXpage
\stoptext
```

With LuaTEX we see this in the page stream:

```
BT
/F1 11.955168 Tf
1 0 0 1 0 0.11949 Tm [<0069003200620069>]TJ
ET
```

and also this in the mapping from index to Unicode, which is object 14, defined by the `ToUnicode` value shown above (I added the characters as comments):

```
3 beginbfchar
<0032> <0065> % e
<0062> <0073> % s
<0069> <0074> % t
endbfchar
```

When we use LuaMetaTEX instead we get:

```
BT
/F1 10 Tf
1.195517 0 0 1.195517 0 0.065717 Tm
[<0001000200030001>] TJ
ET
```

and:

```
3 beginbfchar
<0001> <0074> % t
<0002> <0065> % e
<0003> <0073> % s
endbfchar
```

Thus, where LuaTEX uses the original index in the font (not to be confused with the character's Unicode value, if it has one at all), in LuaMetaTEX, or more accurately with the ConTEXt backend, we start at one and number upwards. This gives smaller files.

The only way to find out what an index is actually referring to is to consult the abovementioned `ToUnicode` vector in the font resource because there we map from index to Unicode. That information is used when you search in a PDF file or cut-and-paste from it. Because glyphs can be unrelated to Unicode, and because multiple glyphs can share the same Unicode slot, the index is what makes a glyph unique.

When a (page from) a PDF file is included in a document LuaTEX will copy the relevant objects to the main file. Of course the page itself is copied (with the page stream making up the content). Copying is also driven by the `Resources` key in the page dictionary. In addition to the `Font` list we've seen above, there can also be an `XObject` array and its entries need to be copied as well. This copying is recursive because the resources themselves can point to objects with resources. It is quite normal in a PDF file to share resources. The rendered glyphs, for instance, come from fonts that contain the shape definitions and basically these are the same, independent of scaling.

Table 1: Comparison of LuaTEX, LMTX, PDF compression, and advanced merging.

			native	`compact=no`	`compact=yes`
LuaTEX	MkIV	compressed	839 KB / .20 sec	731 KB / 0.20 sec	231 KB / 0.22 sec
		decompressed	1784 KB / .20 sec	943 KB / 0.22 sec	426 KB / 0.23 sec
LuaMetaTEX	MkXL	compressed		544 KB / 0.18 sec	147 KB / 0.24 sec
		decompressed		783 KB / 0.16 sec	552 KB / 0.19 sec

The index can be the original glyph index in the font but, because we subset, it can also be a different one, depending on what gets included. This is illustrated in the example above. By default the LuaTEX engine just copies and doesn't worry about what gets copied. However, in MkIV we can load some code that plugs into the LuaTEX backend and is thereby capable of merging fonts from the included PDF (page) with one used in the document. This process is driven by setting the `compact` key in `\externalfigure` to the value `yes`. Here we assume that the references to glyphs in the page stream are the original (or equivalent) indices but this is not guaranteed to be true. We can check a little by comparing the Unicode mapping as well as doing a visual check afterwards, but neither are robust. It still works ok as long we use exactly the same font; essentially, we check the name and when it matches we force the glyph into the current file and use the font reference of main document for the embedded reference instead.

In LuaMetaTEX we do it differently and there are reasons for this. First of all, we have different numbering in the main file and inserted file, so we cannot use the indices directly. In addition, we have to also take care of Type 3 fonts that refer to fonts that we merge (we use these fonts in, for instance, math delimiters). Finally we cannot simply look at the name because we can have a variable font instance that has different axis properties. So we have to be more clever: we need to parse the content streams of the page, XObjects and charprocs to find out what glyphs are referenced and replace indices when applicable. In addition we consult some extra information that is included when ConTEXt did typeset the (to be embedded) file. That information contains a stream index to original index mapping, and also has a variable font recipe if needed. There is also some additional information so that we at least check if we have the same font.

In Table 1 we compare three alternatives. The native inclusion in LuaTEX leaves the work to the engine. When the plugin is loaded, we will use the Lua-based inclusion code which is a bit more clever in sharing objects. In the LuaMetaTEX variant we also copy objects into the main file but there we have no plugin and sharing happens anyway. Here with `compact=yes` we also merge fonts but this time based on parsing the streams. This parsing is more demanding and bumps runtime but is also more rewarding in terms of file size. For completeness we show the results with and without PDF object stream (zip) compression. The need to decompress and compress also has some impact on performance.

In case the slightly slower inclusion in LuaMetaTEX bothers you, there might be some comfort in knowing that the 20 files accumulate to 564 KB and a fresh run takes 49 seconds. When we use LuaTEX the file size total bumps to 748 KB and the initial runtime goes up to 94 seconds. So in the end the LuaMetaTEX-based variant is more efficient.

So, in MkIV there are two reasons for having the plugin. The first is that by sharing common objects we can, for instance, include many pages from the same document with little overhead. For this, compact doesn't need to be active. However when for instance we include many documents we can see that merging fonts does pay off handsomely. In MkXL we already have the first benefit (sharing) by default and here we can also do better by merging fonts. Because that merging is more aggressive you see better numbers in the table for MkXL.

A practical usage scenario is making manuals where we process examples (using ConTEXt buffers) in independent runs so that they are independent from the main document. Of course there is only a gain if these examples share fonts with the main document or with each other. Here is the test case:

```
\startbuffer[common]
    \usebodyfont  [dejavu]
    \usebodyfont  [lucida]
    \usebodyfont  [bonum]
    \setupbodyfont[modern]
    \setupalign[tolerant,stretch]
\stopbuffer
```

We load four different font sets in a common buffer but also use them in the main document:

```
\getbuffer[common]
```

We define two additional buffers that each create a document with two pages. We use different languages because otherwise there is little to merge, as the sample texts use the regular Latin script:

```
\startbuffer[demo-1]
  \start
    \switchtobodyfont[dejavu]\samplefile{ward}
  \stop \blank
  \start
    \switchtobodyfont[lucida]\samplefile{davis}
  \stop \page
  \start
    \switchtobodyfont[bonum] \samplefile{knuth}
  \stop \blank
  \start
    \switchtobodyfont[modern]\samplefile{tufte}
  \stop \blank
\stopbuffer

\startbuffer[demo-2]
  \start
    \switchtobodyfont[dejavu]\mainlanguage[es]
    \samplefile{cervantes-es.tex}
  \stop \blank
  \start
    \switchtobodyfont[lucida]\mainlanguage[sv]
    \samplefile{alfredsson-sv.tex}
  \stop \page
  \start
    \switchtobodyfont[bonum] \mainlanguage[de]
    \samplefile{aesop-de.tex}
  \stop \blank
  \start
    \switchtobodyfont[modern]\mainlanguage[cz]
    \samplefile{komensky-cz}
  \stop \blank
\stopbuffer
```

Optionally we enable compact inclusion:

```
% \setupexternalfigures[compact=yes]
```

Here is the main document:

```
\starttext
  \dorecurse{10}{
    \startTEXpage[offset=1ex]
      \start \switchtobodyfont[dejavu]
             \samplefile{ward}  \stop \blank
      \start \switchtobodyfont[lucida]
             \samplefile{davis} \stop \blank
      \start \switchtobodyfont[bonum]
             \samplefile{knuth} \stop \blank
      \start \switchtobodyfont[modern]
             \samplefile{tufte} \stop \blank
      % #1 is the iterator:
      \setbuffer[#1]#1\endbuffer
      \hbox\bgroup
        \typesetbuffer[common,demo-1,#1]
                      [width=10cm,page=1]
        \typesetbuffer[common,demo-1,#1]
                      [width=10cm,page=2]
      \egroup
      \blank
      \hbox\bgroup
        % these are processed in separate runs:
        \typesetbuffer[common,demo-2,#1]
                      [width=10cm,page=1]
        \typesetbuffer[common,demo-2,#1]
                      [width=10cm,page=2]
      \egroup
    \stopTEXpage
  }
\stoptext
```

In order to get ten times two unique documents to be included, we smuggle an extra buffer into the subsidiary runs. We need to do this because otherwise the hashes of the content of these sub-documents are the same and we'd end up with only two documents and successive inclusions would share these. Of course the first run with fresh buffers will take more runtime because the sub-documents need to be processed (twice in order to get multi-pass activities resolved).

There are a few pitfalls. First of all we have to make sure that we only merge references to the same font. This is often no problem as long as we don't update fonts with different shapes in the same slots but we can assume that the version number is different then. For the application we have in mind, buffered sub-runs or inclusion in related documents that get processed in a short time span, we are probably ok. We can make the check more tolerant or more clever, and might do that in the future. On the average the inclusion is already rather efficient when a few pages from a few documents are used.

A second pitfall is that when we improve the ConTEXt (font) backend we can have better shapes or more precise metrics but because metrics are unlikely to change much in the glyph programs we're probably okay. Even mixing the more efficient so-called compact font mode with normal font mode (not to be confused with compact inclusion) should work out well enough. In case of doubt: trust your eyes or just regenerate the documents involved in the inclusion.

Finally it is worth mentioning that there is a noticeable overhead but if becomes necessary I can optimize the handling of the stream a bit by replacing the more general stream parser by a dedicated one for this purpose.

Let's stress one thing again. Because shared font usage will never be (guaranteed) watertight you do need to check visually. A bad merge will immediately show up by the included image rendering with garbled text. There are some extra safeguards in the MkXL approach that are absent in the MkIV solution which is why the latter is considered an experiment and not loaded by default. I could spend time on it but as we moved on to LMTX (the MkXL LuaMetaTEX combination) it makes little sense.

Does the story end here? Not entirely. Occasional validation requirements have a side effect that some users have to fix old PDF files to suit demands. Let's mention a few issues users run into:

- Embedded so-called Type 0 fonts can lack a `/CIDSet` and/or `CIDToGIDMap` entry that needs to be added.
- A document can refer to external files that are not embedded. Normally these are in `WinAnsi` encoding and page stream indices match encoding indices so we can smuggle these files into the document.
- Font resources can be embedded multiple times with different subsets, likely per page. Different names are used, which complicates matters, but it makes sense to try to merge them.
- Fonts with the same name but in Type 1 as well as TrueType format, both using the original indices, occur in the same document so they can be merged.

We can deal with this quite well if we have the original fonts available. Failures to do this well immediately show up so we can again trust our eyes. In an automatic large scale fix operation we can built in some safeguards.

Then, as we're fixing included pages anyway, we may as well try to conform to standard even better, for instance:

- Instead of gray scales CMYK and RGB colors are used, or one of these color spaces is not handled right and we need to remap colors. It can be a side effect of lazy programming in producer software.
- Extended graphic states are used, for instance for transparencies while there is no transparency actually being used. These can be side effects of producers just emitting as much as they can.
- Irrelevant grouping can be applied to pages and `/XObjects`. In fact, when a validator complains we can just as well get rid of them.

For such things we need to fix the `/Resources` as well as the page content streams so it adds a little more overhead but when we just convert documents it is a one-time effort so a few more milliseconds won't be a big burden.

But discussing these details doesn't really fit in this font discussion so we end by mentioning that we have a framework in place for plugging in fixers of any kind. The approach is to configure what standard to use and what fixes to apply. Only time will tell if this is sufficient. Most users probably never have to worry about it anyway.

◇ Hans Hagen
Pragma ADE

Is a given input a valid TEX ⟨*number*⟩?

Udo Wermuth

Abstract

This article discusses the question of how one can determine if a given string of characters represents a valid number for TEX. A macro that looks and behaves like a Boolean conditional is implemented to answer the question.

1 Introduction

TEX operates with several data types and structures. We all know, for example, characters, numbers, dimensions, skips, token lists, boxes, files, and macros. Such structures are created and manipulated during the text processing and sometimes we need to get information about the currently stored contents. Thus, TEX provides a couple of conditional tests to gain insights; see pages 209–210 of *The TEXbook* [1]. Except for one, all of these tests return a Boolean result, i.e., *true* or *false*, and allow therefore two branches with different text and code. Five tests compare two items and two of them require a relation for the test as they don't test only for agreement of a single common characteristic.

All Boolean conditionals use the same scheme "`\if...` ⟨*true branch*⟩`\else` ⟨*false branch*⟩`\fi`" in which the `\else` and ⟨*false branch*⟩ can be omitted if this branch is empty. The structure itself is *expandable* ([1], p. 213) and the conditionals can be nested as TEX keeps track about the control words `\if...`, `\else`, and `\fi` even if they aren't executed; conditionals are *skippable* ([1], p. 211).

Plain TEX provides the command `\newif` so that users can create new Boolean conditionals ([1], p. 211). The user must set the conditional: Whatever the flag should mean its result must be computed before. The names of these flags must start with `\if` and thus these conditionals match the above scheme.

For example, TEX has no built-in test for the question if a given input string represents a valid number. So one must code a macro for this test and then a user-created Boolean conditional can be set to true or false. Unfortunately, the macro that must be coded turns out to be rather complex.

The TEX FAQ (accessible through TUG's Internet site or `https://texfaq.org/FAQ-isitanum`) contains information about this question. It focuses on short code snippets and so it limits itself to a discussion about "Is the input a not too large signed or unsigned decimal number?" without giving an answer. Similar limitations occur in the code shown on page 361f. of `https://ctan.org/tex-archive/info/apprendre-a-programmer-en-tex/output/apprendre-a-programmer-en-tex.pdf`.

Encodings. TEX knows many encodings for integers ([1], p. 269 and p. 118): decimal, octal, hexadecimal, and as an alphabetic constant. It accepts numbers in the range from $-2^{31}+1$ to $2^{31}-1$. For example, valid numbers are:

`+-"FF` (a *hexadecimal number*; value −255),
`+- -+"FF` (another hexadecimal number; 255),
`-'777` (an *octal number*; −511),
`` `a `` (an *alphabetic constant*; 97),
`+2147483647` ($=2^{31}-1$, the largest number)

whereas `2147483648` ($=2^{31}$) is invalid as it's out of TEX's range.

To check all cases that TEX allows as the encoding of a number and to do a range check is an unnecessary effort for most macro packages with user-supplied integer arguments. Only if the macro offers an interface for receiving integers from external sources does one need to implement TEX's syntax rules.

Contents. This article describes how to implement a TEX macro looking like a Boolean conditional that decides if a given input string forms a valid number. The macro is named `\ifisint`.

Section 2 lists a few expectations that the conditional should fulfill. Section 3 contains the code for `\ifisint`.

2 Expectations a.k.a. goals

Let's state as precisely as possible what we want to achieve with `\ifisint`.

(1) A Boolean conditional should be coded that has a structure similar to the other Boolean conditionals of TEX. It carries the name `\ifisint`. The argument that is tested for being a valid TEX number is delimited by `\Boolend`. Except for this control word the structure is familiar: `\ifisint` ⟨*argument*⟩ `\Boolend` ⟨*true branch*⟩`\else` ⟨*false branch*⟩`\fi`.

The conditional itself doesn't output anything; only the branches might output something.

(2) Any valid number for TEX in any allowed encoding either unbraced and then followed by any number of spaces or between braces and no spaces in front of `\Boolend` is recognized by `\ifisint` and the tokens in ⟨*true branch*⟩ are processed. Any other input makes TEX execute ⟨*false branch*⟩ if it is present.

(3) Of course, not all characters might appear in ⟨*argument*⟩. For example, TEX's comment character, the percent sign, is never part of a number; TEX reports an error if `\Boolend` is commented out. On

the other hand, the input "`2^3`" should throw no error message although the math shifts are missing.

The input "`{2^3}`" can be passed to *any* macro as an argument without error. But without braces "`2^3`" as a single argument throws an error if the argument is not delimited. Entered as ⟨*argument*⟩ to `\ifisint` there should not be an error message.

Not all of TeX's special characters can occur. For example, a single '`{`' starts a group and without an ending '`}`' TeX will report an error.

If a user wants to test any input string without error messages, TeX's special characters need other category codes. Plain TeX provides the macro `\dospecials` that helps in this task; see page 380 of [1]. Furthermore, TeX's special *double-hat notation* ([1], p. 45; "hat" p. 369) doesn't work if '`^`' does not have category 7. The valid encoding of −1 as "`^^m^^31`" is then rejected. So leave the hat character as special if that is possible.

(4) The ⟨*argument*⟩ should receive written-out input. It doesn't make sense to test data stored in, say, a `\count` register to find out if it represents a valid TeX number. But a simple macro that stores a number in its replacement text should be accepted; undefined macros shall be reported.

This makes `\ifisint` different from, for example, `\ifodd`, as this conditional accepts, for example, count registers for its test. With `\ifisint`, code this: `\expandafter \ifisint \the\count`⟨*n*⟩␣ `\Boolend`. Of course, if "⟨*n*⟩" isn't allowed after `\count`, say, because $n > 255$, an error is raised.

(5) The conditional must be skippable, i.e., the following input with nested `\if`s

```
\iffalse\ifisint 117\Boolend\message{A}%
        \else\message{B}\fi
\else\message{C}\fi
```

generates no error message and outputs "C" on the terminal.

(6) It is *not* expected that the new conditional `\ifisint` is expandable.

3 The code for \ifisint

What is a valid integer? This is specified in detail on pages 268–269 of [1]. There are four types of integers in ⟨*normal integer*⟩: a) the ⟨*integer constant*⟩, b) the ⟨*octal constant*⟩ that starts with a right quote, c) the ⟨*hexadecimal constant*⟩ that starts with the ditto mark, and d) the alphabetic constant built from a left quote and a ⟨*character token*⟩. We are not interested in the syntactic quantity ⟨*internal integer*⟩ as it stands for valid integers stored in control words of TeX; see page 271 of [1]. Any type can be followed by an optional space. Moreover, integers can have signs of category 12: '+' and '−'. One can use a chain of signs and separate them by spaces: Page 268 defines ⟨*plus and minus*⟩ and on page 269 it's stated that in ⟨*optional signs*⟩, the signs might be followed by ⟨*optional spaces*⟩.

And what does TeX do if it expects a number but finds none? For an answer we have to look into [2], part 26, "Basic scanning subroutines". Sections 440–446 contain the code for the *scan_int* procedure that reads a number. Here we also find the three error messages that can occur. Section 442 presents the first error message "Improper alphabetic constant", section 445 contains "Number too big", and section 446 includes the code for the third message "Missing number, treated as zero".

The last message tells us what TeX does if, for example, a letter is read but a digit was expected: It recovers by inserting the number 0; nothing is removed from the input. In the first error it happens too, as explained in the help text. TeX uses its largest known integer 2147483647 when it finds a number whose absolute value is too big; again, information from the help text. In this case all digits of the large number are read and digested by TeX.

Analysis. Thus, in essence there are six cases that our new macro must distinguish.

1. TeX reads a valid number; no more input.
2. TeX reads a valid number; more input available.
3. TeX doesn't find a number, uses 0 instead; no more input.
4. TeX doesn't find a number, uses 0 instead; more input available.
5. TeX reads a number that's outside of its range, uses 2147483647 instead; no more input.
6. TeX reads a number that's outside of its range, uses 2147483647 instead; more input available.

Only case 1 is a valid TeX number. When we are able to determine if more input is available then cases 2, 4, and 6 are detectable.

Only the following input strings fulfill case 3: the empty input, '`'`', '`"`', and '`` ` ``'. All can be preceded by any number of signs.

Case 5 remains. To distinguish it from case 1 we must be able to check the infinite number of inputs that represent the largest number of TeX. The number is infinite as we can always add another plus or minus sign and more leading zeros in the input.

What we need is a *canonical form* into which we transform the input. With unsigned numbers only a few forms for case 3 remain. An unsigned number with exactly one leading zero leaves for case 5 only three forms: the largest number with a leading zero in decimal, octal, and hexadecimal notation.

Thus, a couple of comparisons solve the main task if we find answers to the following problems.

A. Find a way to detect if the number is followed by more input.
B. Find a way to construct the canonical form.

Problem A is solved with an assignment of the input string to a count register *inside* an hbox. This box is empty if only a number is input.

```
\setbox0=\hbox{\count255=<input>}%
\ifdim\wd0>0pt % <input> is not a number
```

Problem B is solved with two macros: The first removes the signs and the second leading zeros but also assures that one is present. They use the technique called *tail recursion* ([1], p. 219) to do their job. For example, the following code removes signs.

```
\def\II@rmsign #1{\ifx#1+\else\ifx#1-\else
 \II@endrm#1\fi\fi\II@rmsign}
\def\II@endrm #1\fi\fi#2{\fi\fi#1}
```

If the argument to `\II@rmsign` is '+' or '−', one of the two `\ifx` becomes true, the sign is gobbled as nothing is done in the branch, but at the end the macro is called again. The macro stops if the argument isn't a sign; a trick to shuffle the argument and the `\fi`s is needed, though. To make sure that the macro stops we add a *sentinel*, the letter 'W', to the input. This moves the detection of a bad alphabetic constant from case 3 to the box-width check.

But this solution has a shortcoming. It removes not only signs but also signs in curly braces, while such symbols aren't allowed in a valid number. Well, the validity of the number is determined by other means. We don't care that valid and invalid numbers are mapped to the same canonical form at this stage.

To summarize: We do the following steps in a TeX macro.

Step 1: 1) Remove signs; add sentinel. 2) Test that case 3 is excluded; otherwise return false.
Step 2: Create canonical form.
Step 3: 1) Assign the input to a `\count` register inside an hbox. 2) Test that the box width is the width of the sentinel; 3) otherwise return false (cases 2, 4, 6).
Step 4: 1) Return true if the number isn't TeX's maximum. 2) Otherwise test if the canonical form is TeX's maximum. If yes, return true (case 1). 3) Otherwise return false (case 5).

Note the procedure works with errors that are generated intentionally. As TeX limits the number of errors in a single paragraph to 100 ([2], §76) the macro shouldn't be applied, for example, in a loop.

One task is still open: How to suppress TeX's error messages? We cannot do that but we can switch to `\batchmode` so that the messages aren't displayed on the terminal. The terminal gets a blank line when we switch between `\batchmode` and another mode.

There is a little problem as modes are globally set and in the original TeX we don't know to which mode we must return. The macro uses a configurable parameter; the default is `\errorstopmode`.

My implementation. Note, Step 1 includes the expansion in an `\edef`; see the discussion in section 2, no. 4. And `\Boolend` is given the significance of `\iffalse` to make the macro skippable; see section 2, no. 5. Moreover, `\hbox{\II@font W}` has width 10.2778 pt if `\II@font` represents `cmr10`.

```
\catcode`\@=11 %  use the private prefix ``II@''
\newif\ifII@itis      % main result of the macro
%% helper macros
\def\II@rmsign #1{\ifx#1+\else\ifx#1-\else
 \II@endrm#1\fi\fi\II@rmsign}
\def\II@endrm #1\fi\fi#2{\fi\fi#1}
\def\II@zeros #1{\ifx#1''\else\ifx#1""\else
 \II@cont#1\fi\fi\II@zeros}
\def\II@cont #1\fi\fi#2{\fi\fi\II@hdlzero#1}
\def\II@hdlzero #1{\ifx#10 \else
 \II@xchgfi #1\fi\II@hdlzero}
\def\II@xchgfi #1\fi#2{\fi\ifx#1`\else0\fi#1}
%% constants with the sentinel `W'
\def\II@cfd{02147483647W}%  canonical forms with
\def\II@cfh{"07FFFFFFFW}% W of TeX's max integer
\def\II@cfo{'017777777777W}%    in dec, hex, oct
\def\II@W{W}\def\II@hexW{"W}% all unsigned input
\def\II@octW{'W}% with W for which TeX inserts 0
%% assignments
\let\Boolend=\iffalse \font\II@font=cmr10
\let\IIcurrentmode=\errorstopmode    % CONFIGURE
%% main macro
\def\ifisint #1\Boolend{\II@itisfalse % see S1.2
 \edef\II@digs{\II@rmsign#1W}% S1.1 with 2 \edef
 \edef\II@digs{\expandafter\II@rmsign\II@digs}%
 \ifx\II@digs\II@W\else\ifx\II@digs\II@octW
 \else\ifx\II@digs\II@hexW\else  % S1.2 finished
  \edef\II@cf{\expandafter\II@zeros\II@digs}% S2
  \wlog{=== start ignore}\batchmode\begingroup
   \setbox0=\hbox{\count255=\II@cf
    \xdef\II@val{\the\count255}}%
   \setbox0=\hbox{\II@font\count255=#1W}%   S3.1
   \xdef\II@wd{\the\wd0}%
  \endgroup\IIcurrentmode\wlog{=== stop ignore}%
  \ifdim\II@wd=10.2778pt % \wd of hbox `W'; S3.2
   \II@itistrue \ifnum\II@val=2147483647 %  S4.1
    \ifx\II@cf\II@cfd
    \else\ifx\II@cf\II@cfh
    \else\ifx\II@cf\II@cfo                 % S4.2
    \else \II@itisfalse                    % S4.3
   \fi\fi\fi\fi
  \else \II@itisfalse                      % S3.3
 \fi\fi\fi\fi \ifII@itis}
\catcode`\@=12
```

A few remarks. The second `\edef` for `\II@digs` can be deleted if macro expansion as in section 2, no. 4, is not needed. Currently it's possible to code:

```
\def\mynum{-1234 }\ifisint\mynum\Boolend ...
```

This expansion is performed outside of `\batchmode` so that errors are shown to the user. I did this to avoid misinterpretations if the user enters a faulty sequence and thinks the contents of the macro was tested, i.e., if the user enters something erroneous like this: `\ifisint\maynum\Boolend ...`

The control word `\Boolend` is used in the macro `\ifisint` as delimiter, i.e., `\ifisint` has a *delimited parameter* ([1], p. 203f.). But in the case of delayed execution of `\ifisint`, for example, if the primitive `\expandafter` precedes it, the user must be careful not to execute `\Boolend`.

As mentioned above, `\Boolend` receives via a `\let`-assignment the meaning of `\iffalse`. Thus if TeX executes `\Boolend` it also executes `\ifisint`'s ⟨*false branch*⟩. Therefore, the use of a `\count` register in section 2, no. (4), requires a space between the number of the `\count` register and the delimiter `\Boolend` to avoid the erroneous execution of `\Boolend` that destroys the macro `\ifisint`.

A second `\let`-assignment gives the control sequence `\IIcurrentmode` the meaning of TeX's primitive `\errorstopmode`. A user can change this by another `\let`-assignment so that `\ifisint` returns to the mode that is currently active. (With ε-TeX one can query the current mode and return to it after `\ifisint` has done its work.)

Execution time. TeX needs more time to execute `\ifisint` than it needs to perform `\ifodd`, i.e., the only built-in conditional with a single number. Measurements on my system with my own TeX port in Pascal show that `\ifisint` is ≈ 8.5 times slower than `\ifodd` when the "real" times of 100,000 calls of "`\ifodd 255\fi`" and of 100,000 calls of "`\ifisint 255\Boolend\fi`", with the additional assignment "`\def\wlog #1{}`", are compared.

References

[1] Donald E. Knuth, *The TeXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[2] Donald E. Knuth, *TeX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

◇ Udo Wermuth
Dietzenbach, Germany
`u dot wermuth (at) icloud dot com`

Is a given input a valid TeX ⟨*dimen*⟩?

Udo Wermuth

Abstract

This article discusses the question of how one can determine if a given string of characters represents a valid dimension for TeX. A macro that looks and behaves like a Boolean conditional is implemented to answer the question.

1 Introduction

This text is a follow-on article to [3], which explains how one can decide if a given string of characters is a valid number for TeX; the macro implemented there is named `\ifisint`. In the current article we look at the problem to decide if a given input is a valid dimension for TeX.

This paper also explains the implementation of a macro named `\ifisdim` with the structure known from `\ifisint`. It is essential that a reader of this article has studied [3] as this text often refers to [3] without repeating the introduced techniques.

Contents. This article follows the analysis found in [3] and describes how to implement a TeX macro looking like a Boolean conditional to answer the question of the title; as mentioned above, the macro is named `\ifisdim`. The expectations formulated in [3], section 2, apply to `\ifisdim` accordingly.

Section 2 lists facts about TeX's dimensions that are important to understand `\ifisdim`. Section 3 contains the code for `\ifisdim`.

2 About dimensions

It's too naïve to say that a dimension is a TeX number and a unit; what's correct is that a dimension consists of a *numeric part* and a unit ([1], pp. 270–271). One option for the numeric part is a TeX ⟨*number*⟩, i.e., an integer. All encodings (see [3]) are allowed but not their full range; see below. Another option is the ⟨*decimal constant*⟩, i.e., a number followed by a period or comma and a sequence of digits that builds the fraction. TeX reads all digits that it finds after the period or comma but at most the first seventeen can influence the value of the dimension; see §452 of [2].

TeX respects different traditions of writing decimal constants and therefore accepts two symbols as the separator between the integer part and the fraction. TeX also respects the history of different printing traditions and comes with plenty of units. There are nine ⟨*physical unit*⟩s ([1], p. 57): One can use `pt` (point) and `pc` (pica) from the American stan-

doi.org/10.47397/tb/45-1/tb139wermuth-isdim

dardization in the 19th century or `dd` (didot point) and `cc` (cicero) based on the practice of François-Ambroise Didot in the 18th century. Next, TeX accepts `in` (inch) or units in the metric system: `mm` (millimeter) and `cm` (centimeter). It introduced `bp` (big point) and `sp` (scaled point). Moreover, TeX also knows about traditional units used by typesetters, `ex` (x-height) and `em` (quad width), that depend on the font that's currently in use ([1], p. 60).

The units `ex` and `em` don't belong to the physical units as another parameter is required to determine their values: a font. Here we fix the font to TeX's default font `cmr10` and include both units in the tests of the new macro `\ifisdim`.

The two syntactic quantities ⟨*mudimen*⟩ and ⟨*fil dimen*⟩ carry the word "dimen" in their description but they cannot be assigned to a `\dimen` register. ⟨*mudimen*⟩ ([1], p. 270) must be used with a muskip, which is a glue specification. ⟨*fil dimen*⟩ ([1], p. 271) only occurs in stretch or shrink components of skips and muskips; again it's part of glue specifications. `\ifisdim` doesn't recognize these quantities as valid TeX dimensions.

A valid unit is either one of the nine ⟨*physical unit*⟩s that can be preceded by the keyword `true` to protect it against magnification or the two font-dependent units `em` and `ex`. All units are keywords so that they can be written with category 11 or 12 characters, in upper-, lower-, or mixed-case, and with optional spaces in front of them; see [1], p. 268.

Dimensions are internally represented by TeX in scaled points and TeX uses the unit `pt` if it has to show a stored one. The numeric part of a dimension in scaled points must lie between $-2^{30}+1$ and $2^{30}-1$. Thus, the range is smaller than the one for numbers; see [3]. 1 sp is a very small distance, 65536 sp give 1 pt and that means the maximum decimal constant for the unit `pt` is much smaller than $2^{30}-1$.

Table 1: Ranges for physical units

unit	**max. decimal constant**†	**shown as**
pt	16383.99999237060546874	16383.99998pt*
pc	1365.33333587646484374	16383.99994pt
in	226.70540618896484374	16383.99915pt
bp	16322.78954315185546874	16383.99998pt*
dd	15312.02584075927734374	16383.99997pt
cc	1276.00215911865234374	16383.99995pt
mm	5758.31742095947265624	16383.99997pt
cm	575.83174896240234374	16383.99997pt
sp	1073741823.99999999999999999	16383.99998pt*
Using the `\fontdimen` of `cmr10`:		
ex	3805.32811737060546874	16383.99997pt
em	1638.39749908447265624	16383.99991pt

† With the (at most) seventeen significant decimal places.
* TeX represents this value as $2^{30}-1\,\mathrm{sp} = 1073741823\,\mathrm{sp}$.

The line in Table 1 for the unit `pt` tells us that an infinite number of input strings with this unit are mapped to TeX's largest dimension. Plain TeX sets `\maxdimen` to 16383.99999 pt but TeX shows it as 16383.99998 pt. When TeX has to show a dimension it outputs at most five digits ([2], §103).

Enter the values 16383.99997711181640625 pt, 16322.78952789306640625 bp, and 1073741823 sp to specify `\maxdimen` with the smallest decimal constants for the three units that can do that.

3 The code for `\ifisdim`

A valid dimension is (1) an integer followed by a valid unit or (2) a ⟨*decimal constant*⟩ with a valid unit as described in section 2. Thus, we encounter the three error messages of TeX when it reads an integer as discussed in [3]. The *scan_dimen* procedure in [2], part 26, adds a few new error situations. Sections 456 and 459 contain the message "Illegal unit of measure" once for dimensions and once for ⟨*mudimen*⟩. And section 460 includes the error message "Dimension too large". In total we have to deal with five error messages that TeX might show when it reads a dimension.

The first new error message means that TeX has found (or inserted) a numeric part and expects now one of the valid units — maybe prefixed with the keyword `true`. If it doesn't find one it inserts the unit `pt` to get a valid dimension. The numeric part might have been generated by TeX if it wasn't able to read a number, i.e., TeX might have inserted a zero as described in [3].

The second error message tells us that the combination of numeric part and unit results in a scaled-point value larger than 1073741823 sp. The help text of the error message informs us that TeX throws the input away and uses its largest dimension instead.

Analysis. Let's list all possible scenarios. Several of the following cases appear with and without more input. We know how to handle this from `\ifisint` so it isn't mentioned here again. Only if it is important that no more data is available is it handled as a separate case.

1. TeX reads a valid dimension.
2. TeX doesn't find a numeric part, uses 0 instead, finds a valid unit.
3. TeX doesn't find a numeric part, uses 0 instead, doesn't find a unit, uses `pt` instead.
4. TeX doesn't find a numeric part, uses 0 instead, finds an invalid unit, uses `pt` instead.
5. TeX finds as numeric part a number larger than 2147483647, uses 2147483647 instead, finds a

valid unit, thus the dimension is too large and TeX uses `\maxdimen` instead.

6. TeX finds as numeric part a number larger than 2147483647, uses 2147483647 instead, finds no valid unit, inserts `pt`, thus the dimension is too large and TeX uses `\maxdimen` instead.
7. TeX finds a numeric part and a valid unit but the combination creates a dimension that's too large, uses `\maxdimen` instead.
8. TeX finds a numeric part but no unit, inserts unit `pt` and builds a valid dimension.
9. TeX finds a numeric part but no unit, inserts unit `pt`, the combination creates a dimension that's too large, uses `\maxdimen` instead.

The list is much longer than the one in [3] for `\ifisint`. But a second check shows that several cases can be deleted. Cases 5 and 6 are just special cases of 7 with more error messages. Next, cases 3, 8 and 9 can be avoided if the sentinel (see [3]) is a valid unit, for example, `mm`. This gives a width for an hbox with an assignment [3] that disagrees with the width of the string '`mm`'. And this happens with case 4 too as the invalid unit and the sentinel remain.

We are left with cases 2 and 7 for invalid dimensions. Looking at [3] we are faced in essence with the same cases but this time each case involves units. For example, case 2 excludes input like "`'pt`" that TeX transforms into "`0pt`". So it looks like we have to execute the three tests of [3] together with all valid units. But no, it doesn't hurt to exclude input with invalid units. All we have to do is to check that the input has at most three characters (without the keyword `true`) and starts with '`'`' or '`"`'. The incomplete alphabetic constant is again moved; here it destroys the unit and generates an error.

Case 7 remains. The solution in [3] was to use a list of the canonical forms of the largest integer in all encodings. So here we need a list of the canonical forms for `\maxdimen`. But there seems to be no simple form for the infinitely many input strings that represent `\maxdimen`, as we saw in Table 1.

In order to distinguish case 7 from case 1 we need to do some calculations: We need to determine the input value in scaled points and compare the result against $2^{30}-1\,\mathrm{sp}$. To do that without risk of getting a false result we use three elements.

a. The integer part of the numeric part: `\II@int`.
b. The fractional part plus the unit: `\II@frac`.
c. The unit, maybe prefixed with `true`: `\II@unit`.

The key to success is the fact that in TeX the range for numbers is larger than the range for dimensions expressed in scaled points. Thus the following computation doesn't generate a "Dimension too large" error if `\II@int` is at most as large as the integer part of the maximum decimal constant for `\II@unit` according to Table 1.

```
\dimen255=\II@int\II@unit
\count255=\dimen255 % coerce dimension to number
\advance\count255 by \II@frac
\def\II@calc{\number\count255 }
```

`\II@calc` contains the sum of the number of scaled points of `\dimen255` and `\II@frac`; see [1], p. 270.

How do we get the required information? If we have a dimension, `\II@dist=\II@int\II@frac`, two assignments fill the variables, with an error message if `\II@dist` contains neither a decimal point nor a decimal comma.

```
\afterassignment\II@frac \II@int=\II@dist
```

It's easy to avoid the error by inserting a zero.

It is not much harder to identify the unit. We assign the digits of the fraction — after removing the period or comma — to a `\count` register, leaving the two characters of the unit. Using an hbox we can distinguish the three strings '`pt`', '`bp`', and '`sp`' by their widths. (In general it is not possible to identify all units, for example, the strings '`bp`' and '`dd`' have the same width. But we are only interested in the width if the dimension equals `\maxdimen` and that cannot happen with '`dd`'; see Table 1.) The keyword `true` must also be considered; its width is subtracted if the width of the hbox exceeds a certain value.

There is one problem: Keywords can be written in different ways with lower- and uppercase characters; the characters might even be of category 12. We need to transform them into a canonical form, for example, lowercase letters, to get a unique width.

Thus we need to realize the following procedure.

Step 1: 1) Remove signs; add sentinel. 2) Test that case 2 is excluded. 3) Otherwise return false.

Step 2: Get the parts: 1) `\II@int`, 2) `\II@frac`, and 3) `\II@unit` (as a width).

Step 3: 1) Assign the input to a `\dimen` register inside an hbox. 2) Test that the box width is the width of the sentinel; 3) otherwise return false (includes cases with more data).

Step 4: 1) Return true if the dimension isn't TeX's `\maxdimen`. 2) Otherwise test if `\II@calc` is TeX's `\maxdimen`. 3) If no, return false (case 7). 4) Otherwise return true (case 1).

This procedure works with a lot of intentional errors that TeX reports while `\batchmode` is active. Thus TeX's limit of 100 error messages per paragraph ([2], §76) is reached much earlier than with `\ifisint`.

My implementation. The following private control words — the two declarations `\ifII@itis` and

\II@font, the macros \II@W, \II@octW, \II@hexW, \II@rmsign, and \II@endrm, and the \let-assignments \Boolend and \IIcurrentmode — are reused from the code of [3]. Their code is marked with two comment characters at the right end of the lines in the following code. You might delete this code if you load via \input the file that contains the code of [3].

```
\catcode`\@=11
\newif\ifII@itis      %% reused from \ifisint %%
\def\II@rmsign #1{\ifx#1+\else\ifx#1-\else    %%
 \II@endrm#1\fi\fi\II@rmsign}%  remove signs: %%
\def\II@endrm #1\fi\fi#2{\fi\fi#1}% `+' & `-' %%
\let\Boolend=\iffalse \font\II@font=cmr10 %   %%
\let\IIcurrentmode=\errorstopmode % CONFIGURE %%
\def\II@W{W}\def\II@octW{'W}\def\II@hexW{"W}% %%
%% declarations
\newdimen\II@frac
\countdef\II@cnt=255 \dimendef\II@dim=255
%% helper macros; some use \ifisint's sentinel W
\def\II@bad #1#2#3#4#5#6\II@end{%   numeric part
 \def\II@id{#1W}% is missing but maybe with unit
 \edef\II@X{#6}\ifx\II@X\empty
  \edef\II@X{#5}\ifx\II@X\empty\else\II@Bad\fi
 \else \edef\II@X{\II@mklc#2#3#4W}%
  \ifx\II@X\II@rueW
  \else\ifx\II@X\II@truW\II@Bad
  \else \II@itistrue
 \fi\fi\fi}
\def\II@rueW{rueW}\def\II@truW{truW}
\def\II@Bad{\ifx\II@id\II@W
 \else\ifx\II@id\II@octW
 \else\ifx\II@id\II@hexW
 \else \II@itistrue
 \fi\fi\fi}
\def\II@mklc #1{\if#1pp\else\if#1Pp\else
 \if#1tt\else\if#1Tt\else
 \if#1bb\else\if#1Bb\else
 \if#1ss\else\if#1Ss\else
 \if#1rr\else\if#1Rr\else
 \if#1uu\else\if#1Uu\else
 \if#1ee\else\if#1Ee\else
  \II@endlc#1\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
 \fi\fi\fi \II@mklc}%  `W' and `m' stop \II@mklc
\def\II@endlc #1\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
 \fi\fi\fi\fi#2{\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
 \fi\fi\fi\fi#1}
\def\II@getfrac #1mm\II@end{\global\II@frac=0#1}
\def\II@getcalc{%\II@calc=coerced\II@int\II@frac
 \ifdim\II@unit=26.11119pt %  \II@unit is ``pt''
  \II@dim=\ifnum\II@int<16384
   \II@int\else 0\fi pt
 \else\ifdim\II@unit=27.77786pt %   it is ``bp''
  \II@dim=\ifnum\II@int<16323
   \II@int\else 0\fi bp
 \else\ifdim\II@unit=26.16673pt %   it is ``sp''
  \II@dim=\ifnum\II@int<1073741824
   \II@int\else 0\fi sp
 \else \II@dim=0pt \fi\fi\fi
 \II@cnt=\II@dim \advance\II@cnt by \II@frac
 \edef\II@calc{\number\II@cnt}}
\def\II@point #1#2\II@end{% assign digits of the
 \afterassignment\II@mklc  % fraction to \II@cnt
 \ifx#1.\II@cnt=0#2%
 \else\ifx#1,\II@cnt=0#2%
 \else \II@cnt=0#1#2%
 \fi\fi}
\def\II@getunit #1{\afterassignment\II@hdlfrac
 \II@cnt=#1\relax}
\def\II@rmtrue{\ifdim\wd0>40pt \the\II@dim
 \else \the\wd0 \fi}
%% main macro
\def\ifisdim #1\Boolend{\II@itisfalse     % S1.3
 \edef\II@dist{\II@rmsign#1mm}%             S1.1
 \edef\II@dist{\expandafter\II@rmsign\II@dist}%
 \expandafter\II@bad
  \II@dist\empty\empty\empty\empty\II@end % S1.2
 \ifII@itis                         % S4.1, S4.4
  \wlog{=== start ignore}\batchmode\begingroup
   \setbox0=\hbox{\II@font
    \afterassignment\II@getfrac
     \II@cnt=\II@dist\II@end              % S2.2
    \xdef\II@int{\the\II@cnt}}%             S2.1
   \setbox0=\hbox{\II@font
    \afterassignment\II@point
     \II@cnt=\II@dist\II@end}\II@dim=\wd0
   \advance\II@dim by -17.80559pt % width `true'
   \xdef\II@unit{\II@rmtrue}%               S2.3
   \setbox0=\hbox{\II@font\II@dim=#1mm%     S3.1
    \xdef\II@val{\ifdim\II@dim<0pt-\fi
     \the\II@dim}}%
   \xdef\II@wd{\the\wd0}%
  \endgroup\IIcurrentmode\wlog{=== stop ignore}%
  \ifdim\II@wd=16.66672pt % width ``mm''    S3.2
   \ifdim\II@val=\maxdimen \II@getcalc
    \ifnum\II@calc=1073741823 %             S4.2
    \else \II@itisfalse                   % S4.3
   \fi\fi
  \else \II@itisfalse                     % S3.3
 \fi\fi \ifII@itis}
\catcode`\@=12
```

References

[1] Donald E. Knuth, *The TeXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[2] Donald E. Knuth, *TeX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[3] Udo Wermuth, "Is a given input a valid TeX ⟨*number*⟩?", *TUGboat* **45**:1 (2024), 106–109. tug.org/TUGboat/tb45-1/tb138wermuth-isint.pdf

◇ Udo Wermuth
Dietzenbach, Germany
u dot wermuth (at) icloud dot com

Visualizing the Mandelbrot set with METAPOST

Max Günther

Abstract

With the advance of modern programming languages allowing for parallelized and optimized computation, visualizing the Mandelbrot set has become easier than ever before. METAPOST was not designed for such time-consuming tasks, nevertheless it has surprisingly acceptable performance.

1 Introduction

The Mandelbrot set is a set of points in the complex plane. A point c is part of this set if the sequence z_n defined as $z_{n+1} = z_n^2 + c$ with $z_n, c \in \mathbb{C}$ and $z_0 = 0$ does not diverge to infinity [1, 2]. In practical applications, it is impossible to perform an infinite number of iterations for z_n. For this reason we define a maximum number of iterations $n_{\max}$. In addition, we can stop the iteration as soon as the norm of the position vector $\vec{z_n}$ exceeds a radius of 2, since it can be shown that in this case z_n will eventually diverge to infinity [4].

Herbert Voß has already implemented an algorithm for visualizing the Mandelbrot set in the German journal DTK [3]. My goal was to port his code from Lua to METAPOST, examining how it will perform and what difficulties will arise in this process.

2 Where are the complex numbers?

The first obstacle I encountered was the lack of complex numbers in METAPOST. To be fair, it is pretty unlikely that you will need complex numbers when creating graphics, so nobody bothered to include them. Luckily, the calculation can be performed using basic algebra!

Recall that a complex number $a = x+yi$ consists of both a real part x and an imaginary part y, which can be used to represent the complex number as a point in the complex plane, as shown in figure 1. It is worth noting that this is a two-dimensional coordinate system, which is (of course) supported by METAPOST.

We also need to be able to add and square complex numbers. By looking at the real and imaginary part of a complex number separately, we can calculate $a + b$ and a^2 with ease:

$$(x + yi) + (u + vi) = (x + u) + (y + v)i$$
$$(x + yi)^2 = x^2 - y^2 + 2xyi$$

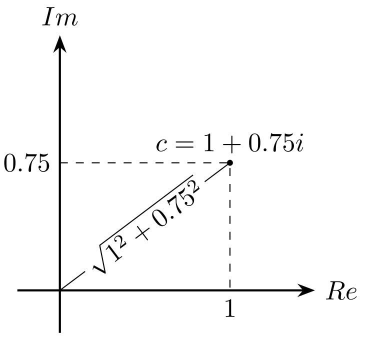


Figure 1: The complex number $c = 1 + 0.75i$ in the complex plane. *Re* is the real axis, *Im* the imaginary axis. The equation notated diagonally uses Pythagoras' theorem for calculating the distance between point c and the origin of the coordinate system.

3 Several arithmetic overflows later. . .

After successfully setting up the innermost loop and checking that the sequence z_n has been calculated correctly, I tried to integrate the two outer loops. During this stage I encountered multiple arithmetic overflows. The mistake was that I carelessly mixed the operators `=` and `:=` in the loop body. An equal sign (without the colon) is the instruction for solving linear equations, *not* the assignment operator. This resulted in unnecessarily constructing a gigantic equation, too huge to be handled by METAPOST.

4 Let it be colorful!

At this point the Mandelbrot set was easy to recognize, but rather dull: each pixel belonging to the set was colored black, the rest was white. To uncover more detail of the Mandelbrot set — especially in the aura — we can use the escape time algorithm. The color of a pixel depends on the number of iterations n completed before the norm of the position vector $\vec{z}$ exceeds the radius of 2. In Voß' implementation, this results in a value between 0 and 255; however, METAPOST expects RGB values between 0 and 1. For that, we can use the following equation:

$$\frac{(n_{\max} - n)}{n_{\max}}$$

5 Optimizing the code

To speed up the computation, I implemented the following optimizations:

1. Extract constants (like `dx` and `dy`) from the loop body to prevent unnecessary computation.
2. Square Pythagoras' theorem, so `sqrt(re**2 + im**2) > 2` becomes `re**2 + im**2 > 4`.
3. Fill the whole image with black and only draw pixels not belonging to the Mandelbrot set.

doi.org/10.47397/tb/45-1/tb139guenther-mandelbrot

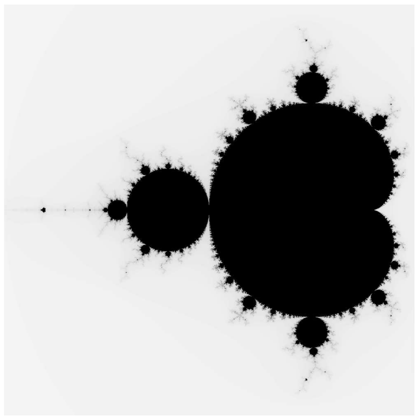


Figure 2: The output of the METAPOST program. The generation of this image with a resolution of 1000 by 1000 pixels took about three minutes on an 11th Gen Intel i7. The following values were used: $x_{\min} = -2$, $x_{\max} = 0.5$, $y_{\min} = -1.25$, $x_{\max} = 1.25$, $n_{\max} = 200$ and $res = 1000$.

The last optimization has the positive side effect of reducing the number of conditional statements needed for drawing a pixel in the correct color.

6 Conclusion

Trying to implement Herbert Voß' algorithm for visualizing the Mandelbrot set was a great experience. After about 3.5 hours of tinkering I finally achieved convincing results. I enjoyed the process and now feel more confident about working with METAPOST in the future.

Figure 2 shows the final image of the "Apfelmännchen", as this detail of the Mandelbrot set is called in Germany, because it is reminiscent of a man rotated by 90 degrees. The source code is displayed in section 7. Feel free to try it out by yourself and play around with the values to explore different parts of the Mandelbrot set.

7 Final code

Save the following code as `mandelbrot.mp` and run it using `mpost mandelbrot.mp`. The resulting image is called `mandelbrot.png`.

```
outputtemplate := "%j.png";
outputformat := "png";

beginfig(1)
  numeric x_min, x_max, y_min, y_max, res;
  x_min := -2; x_max := 0.5;
  y_min := -1.25; y_max := 1.25;
  res := 1000;

  numeric n_max, dx, dy; n_max := 200;
  dx := (x_max - x_min) / res;
  dy := (y_max - y_min) / res;

  fill (0,0)--(res-1,0)--(res-1,res-1)
    --(0,res-1)--cycle;

  for x=0 upto res - 1:
    for y=0 upto res - 1:
      numeric re, im, old_re, old_im, a, b;
      re := 0; im := 0;
      re_old := 0; im_old := 0;

      a := x * dx + x_min;
      b := y * dy + y_min;

      for n=0 upto n_max:
        numeric squared_re, squared_im;
        squared_re := re**2;
        squared_im := im**2;

        re_old := squared_re - squared_im;
        im_old := 2.0 * re * im;
        re := a + re_old;
        im := b + im_old;

        if squared_re + squared_im > 4:
          numeric c; c := (n_max - n) / n_max;
          numeric o; o := 0.95;
          fill (x,y)--(x+o,y)--(x+o,y+o)
            --(x,y+o)--cycle withpen pencircle
            scaled .1pt withcolor (c, c, c);
        fi;
        exitif squared_re + squared_im > 4;
      endfor;
    endfor;
  endfor;
endfig
end
```

References

[1] B. Fredriksson. An introduction to the Mandelbrot set, Jan. 2015. www.kth.se/social/files/5504b42ff276543e4aa5f5a1/An_introduction_to_the_Mandelbrot_Set.pdf

[2] J. Montelius. Generating a Mandelbrot Image, 2018. people.kth.se/~johanmon/courses/id1019/seminars/mandel/mandel.pdf

[3] H. Voß. Chaotische Symmetrien mit Lua berechnet. *Die TEXnische Komödie*, 32(3):51–57, Aug. 2020. archiv.dante.de/DTK/PDF/komoedie_2020_3.pdf

[4] E.W. Weisstein. The Mandelbrot set — from Wolfram MathWorld. mathworld.wolfram.com/MandelbrotSet.html

◇ Max Günther
code-mg (at) mailbox dot org
www.guemax.de

Semi-automated Ti*k*Z directed acyclic graphs in R

Travis Stenborg

Abstract

Directed acyclic graphs (DAGs) are a key visualisation tool in graph theory. Semi-automated generation of Ti*k*Z code for rendering DAGs is introduced. Automatic Ti*k*Z generation via the causalDisco package of the R statistical programming language is proposed. Such easy, rapid DAG generation for LaTeX environments alleviates the need for tedious manual layout of DAG vertices and edges.

1 Directed acyclic graphs

Directed acyclic graphs (DAGs) are a type of mathematical graph structure consisting of *vertices* connected by *edges*. DAGs have two properties that distinguish them from general graphs. Firstly, DAGs have edges with an associated direction defining an order to the vertices (hence, *directed*). Secondly, the edges never define a path wherein the starting vertex of a path is also its ending vertex (hence, *acyclic*).

DAGs have applications in fields such as causal data science [4], computational optimisation [5] and even TeX paragraph aesthetics [7]. Ti*k*Z rendering allows fine tuning of graph presentation, and easy font matching with underlying LaTeX documents.

2 The causalDisco R package

The "causal discovery" R package, causalDisco, can autogenerate Ti*k*Z code to render DAGs from a concise vertex and edge specification. A version (0.9.1) is available from CRAN, but the more recent version (0.9.3) from GitHub addresses rendering bugs. Additionally, causalDisco has Bioconductor package dependencies. A combined download call in R is:

```
BiocManager::install(c("graph", "RBGL"))
github_repo <- "annennenne/causalDisco"
devtools::install_github(github_repo)
```

3 Automated Ti*k*Z from R

Example R code to render a DAG in Ti*k*Z is given below. A seven-vertex graph derived from coral reef ecology [1] was used here.

```
dag_matrix = matrix(
  c(0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0,
    1, 1, 0, 0, 0, 0, 0,
    1, 0, 1, 1, 0, 0, 0,
    0, 1, 1, 0, 1, 0, 0,
    0, 0, 0, 1, 0, 1, 0),
  nrow = 7, ncol = 7, byrow = TRUE)

# Specify matching matrix row and column names.
rownames(dag_matrix) <- c(
  "a_nd1", "a_nd2", "a_nd3",
  "b_nd4", "b_nd5", "b_nd6", "c_nd7")
colnames(dag_matrix) = rownames(dag_matrix)

# Create a temporal adjacency matrix.
model <- causalDisco::tamat(
  dag_matrix, c("a", "b", "c"))

# Render TikZ and copy to clipboard.
causalDisco::maketikz(model, xjit = 0,
  markperiods = FALSE, addAxis = FALSE,
  varLabels = list(
    a_nd1 = "Depth",
    a_nd2 = "\\footnotesize Structural\\\\
            \\footnotesize Complexity",
    a_nd3 = "\\footnotesize Human\\\\
            \\footnotesize Gravity",
    b_nd4 = "MPA",
    b_nd5 = "\\footnotesize Fishing\\\\
            \\footnotesize Pressure",
    b_nd6 = "\\footnotesize Reef Fish\\\\
            \\footnotesize Biomass",
    c_nd7 = "\\footnotesize Coral\\\\
            \\footnotesize Cover")
  )
```

By default, causalDisco generates \small vertex labels. To better balance the size of graph vertices with multiline vs. single line labels, judicious label adjustment via \footnotesize was made.

4 Finishing touches

There is no shortage of learning material for Ti*k*Z beginners [2, 6, 8, 10, 11, 12, 13, 14, 15, 16, 17]. Assuming Ti*k*Z basics are familiar, the causalDisco manual recommends the following Ti*k*Z preamble.

```
\usepackage{tikz}
\usetikzlibrary{arrows, arrows.meta,
  automata, backgrounds, shapes, snakes,
  petri}
\usepackage{pgfplots}
```

The example causalDisco code copies Ti*k*Z commands to the system clipboard. They should be pasted into a Ti*k*Z \begin{tikzpicture} and \end{tikzpicture} block. The compiled results generate *naked nodes*, i.e. nodes without any encapsulating boundary. Here however, additional Ti*k*Z shape calls, shown below, were manually added to nodes to encircle them (hence, *semi-automated*).

```
\node (1) at (0,1) [shape=circle,draw]
      {Depth};
```

The end result is given in Figure 1.

doi.org/10.47397/tb/45-1/tb139stenborg-dags

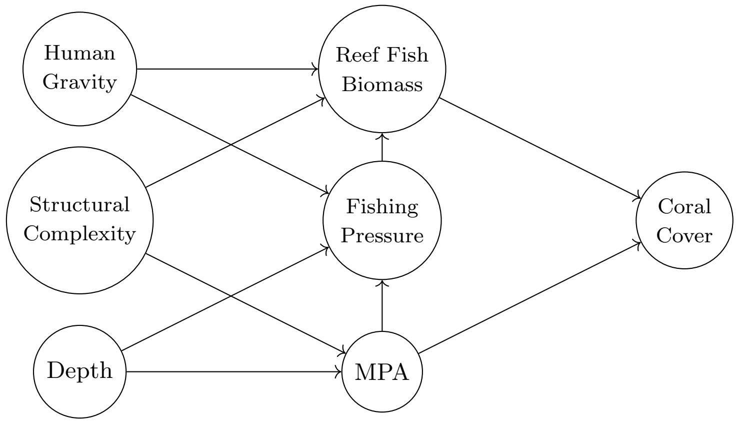


Figure 1: Directed acyclic graph visualising the causal structure of the influence of marine protected areas (MPAs [9]) on reef fish biomass. Adapted from an example in coral reef ecology [1]. *Human gravity* measures the human population near a reef, divided by the square of the time it takes to travel to that reef [3].

Acknowledgements

This work was supported by the Australian Research Council Training Centre in Data Analytics for Resources and Environments (project ICI9010031).

References

[1] S. Arif, M.A. MacNeil. Utilizing causal diagrams across quasi-experimental approaches. *Ecosphere*, 13(4):e4009, 2022.

[2] I. Borja. An introduction to automata design with TikZ's automata library. *TUGboat* 44(1):102–107, 2023. `doi.org/10.47397/tb/44-1/tb136prado-automata`

[3] J.E. Cinner, E. Maire, et al. Gravity of human impacts mediates coral reef conservation gains. *Proc. Natl. Acad. Sci. USA*, 115(27):E6116–E6125, 2018.

[4] G. Gao, B. Mishra, D. Ramazzotti. Causal data science for financial stress testing. *J. Comput. Sci.*, 26:294–304, 2018.

[5] J.L. Gross, J. Yellen, P. Zhang. *Handbook of Graph Theory, 2nd Edition*. Chapman and Hall/CRC, 2013.

[6] G. Grätzer. *More Math Into LaTeX*. Springer, Cham, 5th ed., 2016.

[7] Y. Haralambous. TeX as a path, a talk given at Donald Knuth's 80th birthday celebration symposium. *TUGboat* 39(1):8–15, 2018. `tug.org/TUGboat/tb39-1/tb121haralambous-knuth80.pdf`

[8] S. Kottwitz. *LaTeX graphics with TikZ*. Packt, Birmingham, 2023.

[9] D. Laffoley, J.M. Baxter, et al. Ch. 29: Marine protected areas. In *World Seas: An Environmental Evaluation*, pp. 549–569. Elsevier, second ed., 2019.

[10] C. Maggi. The DuckBoat: The Morse code of TikZ. *TUGboat* 39(1):21–26, 2018. `tug.org/TUGboat/tb39-1/tb121duck-tikz.pdf`

[11] C. Maggi. The DuckBoat: You do not need to be Neo to cope with a TikZ matrix. *TUGboat* 41(1):20–25, 2020. `tug.org/TUGboat/tb41-1/tb127duck-matrix.pdf`

[12] A. Mertz, W. Slough. Graphics with PGF and TikZ. *TUGboat* 28(1):91–99, 2007. `tug.org/TUGboat/tb28-1/tb88mertz.pdf`

[13] A. Mertz, W. Slough. A TikZ tutorial: Generating graphics in the spirit of TeX. *TUGboat* 30(2):214–226, 2009. `tug.org/TUGboat/tb30-2/tb95mertz.pdf`

[14] T. Stenborg. A TikZ rendering of the Arecibo message. *TUGboat* 44(3):375–377, 2023. `doi.org/10.47397/tb/44-3/tb138stenborg-arecibo`

[15] M.R.C. van Dongen. *LaTeX and Friends*. Springer, Berlin, 2012.

[16] Z. Walczak. Graphics in LaTeX using TikZ. *TUGboat* 29(1):176–179, 2008. `tug.org/TUGboat/tb29-1/tb91walczak.pdf`

[17] K. Wolcott. Three-dimensional graphics with PGF/TikZ. *TUGboat* 33(1):102–113, 2012. `tug.org/TUGboat/tb33-1/tb103wolcott.pdf`

◇ Travis Stenborg
Sydney, Australia
ORCID 0000-0002-2693-9628

Nodes and edges with METAPOST: The MetaGraph environment

Federico García De Castro

1 Introduction

The aim of this article is to present MetaGraph, a set of METAPOST macros and utilities developed over the last couple of months as an open-ended environment for drawing *graphs* (in the sense of "nodes and edges"), and intended to complement external graph analysis engines with the versatility of programmatic formatting.

After a quick glance at MetaGraph's capabilities through three demo graphs, the sections below offer a general description of the system, highlighting the *data vs. procedure* approach that makes it different from the plotting routines typically available in those external engines — with a special mention of Ti*k*Z — and offering a general 'feel' for what this approach entails and permits.

All graph-related techniques and terms mentioned here are used simply for illustration purposes, and the details of what they mean in graph theory or analysis do not matter much. What the 'degree' of a node illustrates in this or that example could just as well have been illustrated by its 'k-core number', its various 'centrality' measures, or any such node attribute — I will therefore not be discussing these concepts in any depth. Similarly, I will not go into much detail regarding METAPOST's general syntax — knowledge of METAPOST is surely a good asset for using MetaGraph, but I don't think it's a pre-requisite: the operations shown here should be good base analogs for anyone potentially interested in using the system, even without prior METAPOST experience. I am happy to share the source code.

1.1 Three demo graphs

The graph in Figure 1 is a so-called 'random geometric' graph with 200 nodes and 860 edges, drawn with *a*) two kinds of node marker (• and ∘) and *b*) "lassos", that highlight two particular features of the graph (resulting from two particular graph analysis techniques).[1]

Figure 2 is a graph made from a much larger set of data, but from an abstract point of view it is

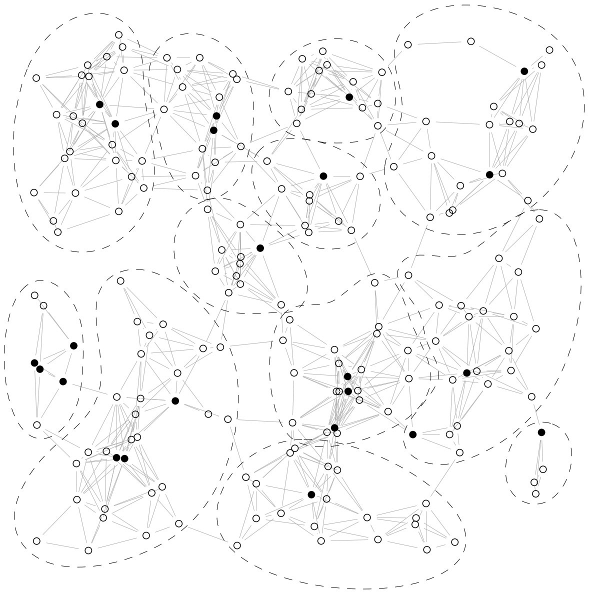


Figure 1: A random geometric graph, 200 nodes and 860 edges

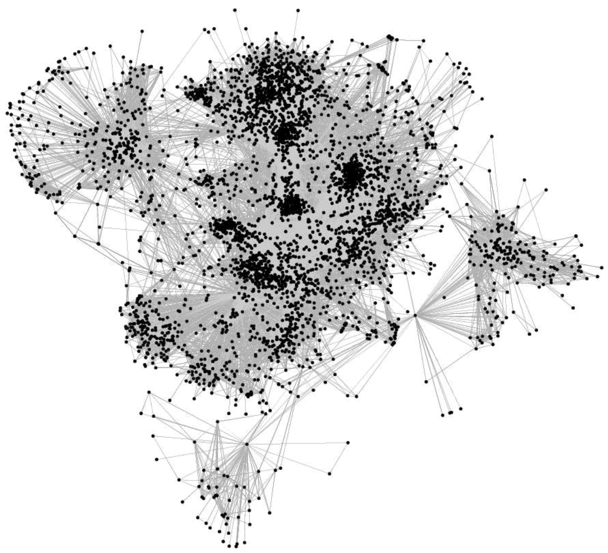


Figure 2: A region of Facebook, 4,039 nodes and 88,234 edges

essentially the same kind of object: a set of name-less nodes and direction-less edges.[2] The figure itself is a much plainer representation of the graph — just black node markers and grey edges — but there is nothing to prevent the kind of lassoing, conditional formatting, or other graphic treatment that was done on the first graph. The main reason to include Figure 2 here was to test the limits of MetaGraph; as it turns

[1] For graph theory folks: The •-nodes are nodes with 'revcore dependency' equal to 0 (meaning that they are local centers of density according to 'reverse core decomposition'); the groups of nodes lassoed in the figure are the node communities yielded by the 'Louvain community detection' algorithm. Both things were computed in a graph-analysis engine, and fed to MetaGraph in a 'data file', as will be explained in more detail below.

[2] These two graphs come from the documentation of the Python library 'Networkx' at `networkx.org/nx-guides/content/exploratory_notebooks/facebook_notebook.html` and `networkx.org/documentation/stable/auto_examples/drawing/plot_random_geometric_graph.html`.

doi.org/10.47397/tb/45-1/tb139garcia-metagraph

out, even its 88,234 edges are far from exhausting METAPOST's capacity (see section 3). The limit on how large a graph MetaGraph can deal with is likely to be practical rather than computational.

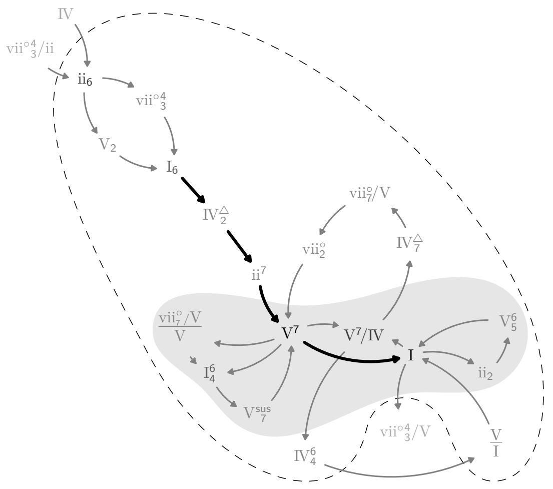


Figure 3: A directed graph of the harmonies of the first prelude in J.S. Bach's *The Well-Tempered Clavier*

The third demo graph is different in a couple of deeper senses. It is a 'directed' graph — hence the arrows in Figure 3 — and its nodes have names — hence the node labels instead of markers. The visualization in the figure features *a*) both a lasso and a shaded region — for METAPOST the two things are not too different, the former resulting from '`draw dashed evenly`', the latter from '`fill`' — and *b*) conditional formatting on the nodes and edges, according to node 'degree' and edge 'weight' (details in section 2.4).

It is this kind of graph that I have been working on in the context of music-theory research, and it is the need for chord-ciphers such as $\frac{\text{vii}^{\circ}_{7}/\text{V}}{\text{V}}$, annoying outside of TEX, that led me to *a*) explore the possibility of writing some useful macros; *b*) realize how adequate METAPOST is for these matters; and *c*) share the news. I imagine the truly easy handling of nodes as LATEX expressions has wider appeal ('H_2SO_4', etc.).

1.2 MetaGraph, Ti*k*Z, etc.

To be sure, most relevant external environments can handle LATEX, 'importing' it or its output into their workflow. In particular, Ti*k*Z (which handles TEX natively, of course) has a truly sophisticated library for graphs, which includes even algorithms that are not present in all graph analysis engines — as well as facilities to implement new ones. Ti*k*Z offers options for the style, placement, coloring, and even animation of nodes, as well as for (several) ways of connecting them with edges. In addition, Ti*k*Z is well integrated in the graph-analysis landscape, and most engines have a backend to export their graphs as Ti*k*Z code, just as they can typically export them as matplotlib plots.

There is, however, little overlap between Ti*k*Z and MetaGraph — in fact, just as little as there is between MetaGraph and matplotlib. MetaGraph is not intended to be a self-contained unit implementing a comprehensive syntax for every possible node- and edge-drawing need and option. If anything, it pursues the opposite: as open-ended an environment as possible, where the graph (its data, essentially) is little more than 'set up' for later straightforward drawing through METAPOST. The exact nature of this approach, its possible benefits and utility, and its difference with Ti*k*Z, will perhaps be clearer after reading or even only surveying the sections below.

1.3 What is MetaGraph?

1.3.1 A set of METAPOST macros

In the strictest sense, MetaGraph is simply a set of METAPOST macros: high-level shortcuts that expand into the plain METAPOST constructions that draw edges, add node labels, etc.

For example, the `allnodes` macro expands to '`0 upto last_node`', which one can use freely to loop, '`for node = allnodes`'. Another macro, `addlabel`, expands to

`addto currentpicture also label`⟨*node-id*⟩

where `label` in turn stands for '`nlabels`⟨*node-id*⟩ `shifted pos`⟨*node-id*⟩' — and so on: `pos` is itself a macro (more on this in section 2.1).

There are also more complicated routines. When drawing an edge (`arrowedge` or `lineedge`), the system checks on the labels of the two nodes involved, so that the edge is drawn starting and ending on the corresponding intersection points, depending on various values such as `labelpadding`, `edgeangle`, etc. There is a family of utilities — `leftoflabel`⟨*node*⟩, `belowrightoflabel`⟨*node*⟩, `abovelabel`⟨*node*⟩, and others — that return the coordinates of the requested point, so that they can be used, for example, in the drawing of lassos.

1.3.2 A graph-drawing *system*

Through macros like these — collected in the METAPOST file `metagraph.mp` — MetaGraph acts as an interface between the actual node and edge data and the METAPOST operations that are most commonly useful to draw the actual graph out of those data. In other words: *given the node and edge data*, there are macros in MetaGraph that provide high-level, graph-oriented utilities to have METAPOST produce the images.

But these 'node and edge data' are expected to come from somewhere else — typically an external graph analysis engine — and they must follow certain conventions in order to be understood by `metagraph.mp`. In this wider sense, MetaGraph is actually a *system* for drawing graphs, consisting, at the time of writing, of *a*) the macros, and *b*) the conventions that need to be followed to provide the raw data of the graph. Eventually, MetaGraph should/will also include *c*) documentation, and *d*) backends for the most common graph-analysis engines.[3]

1.3.3 Dependencies and work flow

One of the good things about METAPOST (over its inspiration, METAFONT) is that it handles TEX natively. It may still give some installation/configuration trouble, since it needs to be pointed to the actual TEX engine used (plain? LATEX? other?); but it is a primitive feature of METAPOST.

One needs also communication in the other direction: from METAPOST into TEX. Since METAPOST produces generic image files (PNG or SVG in addition to PostScript), this is covered by the existing methods to import images into TEX documents.

Of note in this connection is also LuaTEX/LuaLATEX, and its METAPOST library/package luamplib, that takes full care of the communication in both directions (with luamplib, a METAPOST picture is just a TEX box). If one uses Overleaf, then there is no hassle in ensuring that the different tools and formats are well installed, mutually aware, etc.

1.3.4 Under construction

It must be said that MetaGraph is under construction. It is fairly operational — the graphs in these pages were all created with its existing routines — but there are points of syntax still undergoing improvement, a lingering low-levelness, and a chance that further routines may be identified and implemented as I myself become familiar with its capabilities. (The lassoing routines, for example, were entirely developed during and because of the writing of this article — they were only an intuition before I ran into the actual occasion to develop them.) More damaging yet, I have yet to compile a complete, even complete enough, reference manual.

MetaGraph will sooner or later make its way to CTAN, and, in the form of backends, to Networkx and other external graph tools. In the meantime,

[3] At present, there exists the one I wrote for the combination of Networkx and pandas that I use for graph analysis in Python.

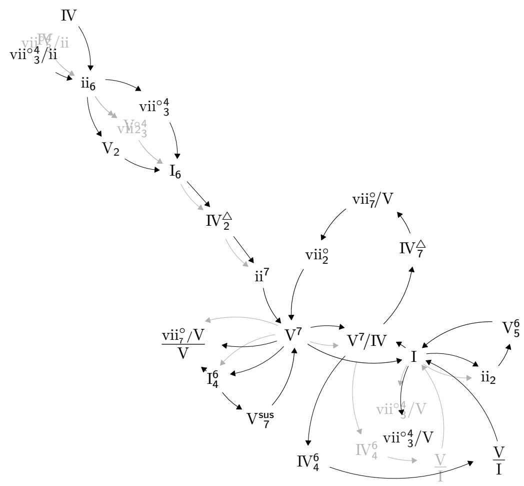


Figure 4: Tweaking of the Figure 3 graph of Bach's prelude

if there is any interest, I am happy to share both the macro definitions and the source code for the illustrations here.

2 Data vs. drawing

The key fact about MetaGraph as a system is that it keeps a complete separation between the graph data and the drawing operations. The first line of code in a MetaGraph graph is usually

```
  input metagraph;
```

(the library of METAPOST macros), while the second is, for example,

```
  input rgdata;
```

(the data file for the random graph of Figure 1).

The two files are completely independent; they are each useless by themselves, but they know exactly nothing about each other.

This way of proceeding preserves two things: *a*) a general programming environment, where the data is assigned for its own sake, and can be used in any way by any METAPOST routine; and, as a result, *b*) direct access to individual graph nodes and edges, for manual or programmatic manipulation, inside or outside other procedures, in the same figure or in a different one.

Figure 4, for example, shows the graph of Bach's prelude (the one in Figure 3), laying a 'tweaked' version in black ink over a lighter-shade layout (yielded by the Fruchterman-Reingold algorithm, one of the 'force-directed' algorithms for laying out graphs). The tweaks include the nudging of certain nodes, to avoid collisions present in the original layout in some cases — in others simply to accommodate the lasso of Figure 3. Some edges are also tweaked. Curved arrows are usually good for directed graphs — they make bi-directed edges easier to see — but the default outgoing angle, calculated blindly with respect to the source node, often creates less-than-satisfying layouts.

Theoretically, these tweaks are of course less than crucial; but they provide a good illustration of the benefit of direct access to individual nodes and edges. Three kinds of commands were used to make the tweaks in Figure 4 (only one instance of each is listed):

```
npos[IV] := % avoid collision
    npos[IV] rotated -3 + (.05, .05);
set_angle((IVj2, ii7), 0); % straight arrow
flipangle(IV, ii6); % flip the arrow
```

This kind of manipulation is much less straightforward in a system where both the data of nodes and edges and the global graphic options (arrows on/off, edge angle such and such, etc.) are issued in the same line.[4]

2.1 Nodes, indices, and arrays

What we have been calling the 'node data' is simply a series of METAPOST arrays. A 'node', for MetaGraph, is just the index number that points to the node's place in those arrays (naturally it is the same in all of them).

In other words, the node's index is the ⟨*node-id*⟩ in expressions like `addlabel`⟨*node-id*⟩, for example. As mentioned, this particular macro expands to the METAPOST line '`addto currentpicture also label`⟨*node-id*⟩', passing ⟨*node-id*⟩ along to `label`. The latter will pass it on in turn as `pos`⟨*node-id*⟩ and `nlabels`⟨*node-id*⟩:

- The data file includes the position of the nodes in the array `npos`⟨*node-id*⟩. Graph-analysis engines typically issue position information without thinking of a particular point size (usually normalizing to the first quadrant, or to the unit circle, etc.), and therefore the values in `npos` need to be scaled; this is what `pos`⟨*node-id*⟩ does.
- Unlike node positions, node labels are not required by MetaGraph. If a graph does have node labels, its data file will have created the array `nlabels`, where MetaGraph will be able to look up each node's LaTeX expression. In label-less graphs, `nlabels` does not exist, and MetaGraph will simply supply the default node image (or whatever the user may define for it).

All throughout the process, ⟨*node-id*⟩ picks out the information for the particular node at hand from each of the data arrays.

[4] One would have to issue three subgraphs: one with the straight edges, one with the clockwise edges, and one with the counterclockwise edges.

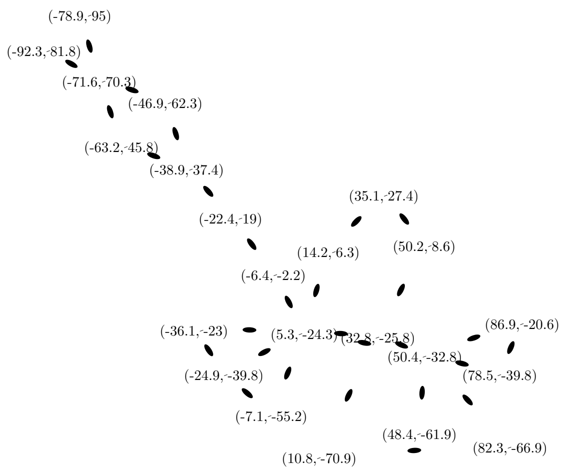


Figure 5: A funny view of the Bach prelude's graph, with the (rounded) raw node positions as 'labels', and blobs — rotated in the direction from one node to the other — in the midpoint of each edge.

2.2 General programming environment

The preceding is the fate of ⟨*node-id*⟩ through the `addlabel` process; but rather than illustrating the workings of that particular macro, the point here is to stress that we are performing general, open-ended programming on raw values with no intrinsic semantics. Adding the label will be the most common use for the node position value; but *at all times* this is just a `pair` variable like any other, and it can be used as such. We can use it — as in Figure 5 — as the label itself, or find the mid-point between two of them, or find out, for whatever purpose, `if angle(npos`⟨*node-id*⟩`) > ctcl_angle`.

2.3 The lassos

Interestingly, the general-purpose nature of METAPOST as a programming language makes MetaGraph a suitable environment in which to implement graph-theoretical techniques. It is relatively easy, for example, to extract a graph's 'line-graph' (a version of the graph whose nodes are the original graph's edges, connected iff they originally share a node), or perform '*k*-means' or 'DBSCAN' clustering. But these techniques are, after all, likely to be available in whatever graph-analysis engine one is using in connection with MetaGraph.

This is *not* the case with lassoing. Figure 6 zooms in on one of the communities (just SE of the origin) of the random graph in Figure 1. This particular lasso (the fifth place in an array of METAPOST paths created for the purpose, called `comms`) was produced by the following code:

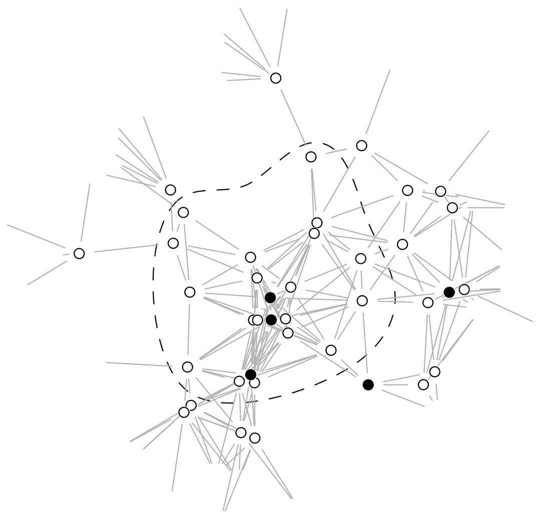


Figure 6: A close-up of one of the communities in the random graph of Figure 1

```
comms[5] = 1/2[pos159, pos152]..tension 1.5
    ..1/2[pos129, pos40]..tension 1.3
    ..1/2[pos103, pos158]
    ..1/2[pos137, pos90]
    ..1/2[pos22, pos14]
    ..1/2[pos163, pos4]
    ..abovelabel168{left}
    ..1/2[pos152, pos168]
    ..cycle;
```

Except for '`abovelabel`' (a MetaGraph macro that finds the point above the label of a given node), this is all plain METAPOST syntax:

- The '`..`' connector creates a (Bézier) curved path between two points. ('`--`' would create a straight line.)
- The `tension` t modifier (with $t = 1$ by default) is one of the ways to control the Bézier curves thus created. (Another is '`controls` c_1 `and` c_2', to specify explicitly the two control points.)
- The convenient $\frac{a}{b}[p_1, p_2]$ construction finds the coordinates of the point that lies $\frac{a}{b}$ of the way from p_1 to p_2.
- '`{left}`' tells METAPOST that the path should travel left at that particular point. (Naturally, one can direct a path `{right}` instead, or `{up}`, or `{down}`, or indeed any `{`⟨*custom vector*⟩`}`; if one wants a particular angle θ, '`dir(`θ`)`' provides the corresponding vector.)

This path (and the others) was designed by visual inspection and trial-and-error. From a run of the 'Louvain community detection algorithm' in Networkx I knew the sets of nodes (i.e., node indices) in each community, and I was looking at a separate version of the graph with the node indices as labels, so that I could locate each community and design its lasso. There is essentially no algorithmic way of lassoing arbitrary sets of nodes, but direct access to node information for varied uses makes it possible to generate paths quickly and robustly.

Incidentally, the clipping of the graph to zoom in on this particular region was also made through node information: for Figure 6, I instructed MetaGraph to include only those nodes and edges that are less than 50 points removed from the center of the lasso (calculated ahead of time as '`lassoctr`'):

```
for edge = alledges:
  if (length(pos[source(edge)]-lassoctr) < 50)
  or (length(pos[target(edge)]-lassoctr) < 50):
    lineedge(edge) withcolor .7white;
  fi;
endfor;
for node = allnodes:
  if length(pos[node]-lassoctr) < 50:
    addlabel[node];
  fi;
endfor;
```

2.4 Node (and edge) attributes

So far we have only referred to node positions and node labels as part of the data that MetaGraph expects to find in the graph data file. Other pieces of information are also required contents of the data file; we will discuss those in section 2.6. Here we shall deal with the possibility of adding and manipulating custom data.

Many attributes of nodes and edges are often relevant for the way a graph is represented graphically (since they are often what needs to be shown). This includes both intrinsic attributes like weights, degrees, etc., and information resulting from global graph analysis — things like k-core and -truss numbers, various 'centrality' measures, etc.[5]

Just like positions and labels, all these attributes are raw data for MetaGraph. But while the arrays of positions and labels are node-specific, one-by-one arrays of n values (n being the number of nodes), most other attributes are encoded in the data file as arrays of *lists* — one list for each value v of the attribute; paraphrasing, lists like "the nodes of attribute *foo* equal to value v are: these and these".

This is much more efficient and straightforward to use than node-by-node arrays, since most metrics

[5] The shaded region in Figure 3 (page 118), for example, is the '3-truss' of the graph, where every edge is part of at least $(3-2)$ triangles, and the lasso its '2-core', where every node has at least 2 edges.

The 'degree' of a node is the number of edges incident upon it; edge 'weights' usually encode empirical observations of a kind or another — in the harmonic-behavior graphs like that of Bach's prelude, they represent the frequency of the progression between two chord-nodes.

group nodes and edges, rather than having single values for each: rather than looping over nodes and checking for their *foo* value, the program (or the user) can loop over the list of *foo*-valued nodes.

The full specification of an attribute `foo` in MetaGraph consists of:

- The pluralized `foos`, which holds a list of the possible values the attribute takes.
- Single-value variables `foo_min` and `foo_max`.
- Explicit lists for each value, `foo_values`⟨*value*⟩, whose contents is a list of the corresponding indices.
- A macro `foo` that expands these lists.[6]
- A prefix: all of the above are actually `nfoo` or `efoo`, according to whether they are node or edge attributes.

For example, the attribute 'frequency' of the nodes in the graph of Bach's prelude is the number of occurrences of each particular chord in the piece. It can be represented as follows in the graph's data file:

```
def nfreqs = 1, 2, 4, 5 enddef;
def nfreq = scantokens nfreq_values enddef;
string nfreq_values[]; % declares array
nfreq_values[1] =
    "ii2, V65, ii6, V2, viid43, IVj7, sivd7,
    viid2, sivd7oV, sid43, sivd43";
nfreq_values[2] =
    "IV, I6, ii6, IVj2, ii7, I7, I64, Vsus7,
    IV64, VoI";
nfreq_values[4] = "I";
nfreq_values[5] = "V7";
nfreq_min = 1; nfreq_max = 5;
```

Note that the lists are not given in terms of node index numbers, but rather expressions that resemble the labels of the nodes: 'ii2', etc. This will be addressed in the next section.

Armed with these lists, conditional formatting based on a given attribute is straightforward. The original view of the graph (Figure 3), for example, makes the grey level and the width of the edges depend on the attribute `efreq`:

```
for freq = efreqs: % freq values present
  for edge = efreq[freq]: % edges of each freq
    arrowedge(edge)
      withpen pencircle scaled ((freq/5)*mm)
      withcolor (.5*(efreq_max-freq))*white;
endfor; endfor;
```

The shading of the nodes, in turn, is a function of another node attribute present in the data file — the node degree:

```
for dg = ndegrees: % degree values present
  for node = ndegree[dg]: % nodes of each dg
    addlabel[node]
      withcolor (.8 - dg/ndegree_max)*white;
  endfor;
endfor;
```

Being raw data, the attributes can just as well be used independently of the graph itself — for example, to produce attribute distribution diagrams, common in graph analysis. Here is the relevant loop for the node degrees of the random graph of Figure 1:

```
pickup penrazor xscaled 6pt;
for dg = ndegrees:
    count := 0;
    for nodes = ndegree[dg]:
        count := count + 1;
    endfor;
    draw (6*dg, 0) -- (6*dg, 2*count);
endfor;
```

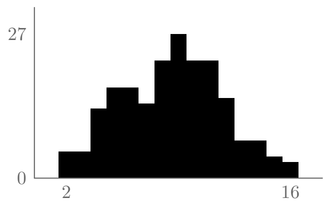


Figure 7: Degree distribution of the graph in Figure 1

2.5 Nodes, labels, and aliases

We have seen that the MetaGraph's ⟨*node-id*⟩s are numbers (section 2.1). But nothing prevents one from assigning these numbers to a series of named variables, which then function essentially as node aliases.

This is particularly relevant in graphs with labeled nodes. The data file of such a graph should contain ASCII-friendly versions of the node labels, as variable names equaling each node's index. That is the case with the graph of the Bach prelude, whose data file creates these aliases as its first order of business:

```
% Nodes
I = 0;
IV = 1;
...
viid43 = 16;
viid2 = 17;
...
```

As we have seen, even the data file uses these variables (rather than the indices) to define the lists of node attributes. These aliases are in fact *entirely* equivalent to the indices; both the program and — more to the point — the user can use them to operate on the nodes (flipping or setting the angle of a

[6] Lists are not a METAPOST data type; they are implemented as string variables, using the primitive `scantokens` to process them; `foo` takes care of this.

curved edge or nudging the position of the nodes, as we did in Figure 4, or manually drawing edges beyond those included in the data file, etc.) without ever needing to care about the otherwise meaningless index numbers.

It is quite fortunate then that METAPOST's variable name conventions allow for monstrosities like `viid43ofii`, or possibly `H2SO4`, and of course benign things like `a`. (The latter is not so benign after all; before I knew better, I was *repeatedly* perplexed by some strange graph-drawing behaviors, only eventually to find that a '`for i`' loop had been clashing with an 'i' that was one of my nodes.)

As long as it starts with a letter, virtually any alphanumeric string is a valid METAPOST variable. It may seem inconvenient to have to refer to $\frac{\mathrm{vii}^{\circ}_{7}/\mathrm{V}}{\mathrm{V}}$ through the all-ASCII name `viid7ofVoV`. But the concession does not come at the MetaGraph stage: already in the graph-analysis engine, if one needs to access nodes individually — to set or get their attributes, for example, or to read out a list of nodes resulting from some clustering operation — then code-friendly names are all but required. Since the engine will then already know these aliases, it can easily add them as variable assignments in the data file. The same names will be available to use at the drawing stage with MetaGraph if they all start with a letter.

2.6 The data file

What attributes to include in a MetaGraph data file is almost entirely discretionary; only a few pieces of information are required. One of them, as mentioned, is the node positions, that go in the `npos` array; another is the `last_node` index, necessary for MetaGraph to loop over all the nodes.

Not mentioned so far is an addition array required by MetaGraph: the list of each node's (outgoing) neighbors, `nodes_to`.

It may not be immediately obvious why this should be required. On the one hand I can report that I have often found myself needing that information for particular graph drawings; on another, it opens up the possibility of having MetaGraph extract subgraphs — although this involves multiple levels of nested loops, and at some point it is better to have the external graph-analysis engine do the work and produce a separate data file. Mostly it has to do with implementing edges in an efficient way, as we will see in the next section.

A minimal data file will then be something like this (coming from `rgdata.mp`, the data file for the random graph in Figure 1):

```
last_node = 199;
% npos (pair)
pair npos[]; i_ := -1;
for value = (0.585, 0.229), (0.722, 0.533),
            ...
            (0.398, 0.516), (0.056, 0.328):
    npos[incr i_] = value;
endfor;
% nodes_to (string)
string nodes_to[];
    nodes_to[4] = "1";
    nodes_to[7] = "5";
    ...
    nodes_to[199] = "167, 13, 81, 50, 23";
```

Then the file may go on to set up the non-required attributes. For example, the node degree attribute in `rgdata.mp` is given by:

```
% ndegree (numeric)
string ndegree_values[]; % declares the array
def ndegrees = 2, 3, 4, 5, 6, 7, 8, 9,
               10, 11, 12, 13, 14, 15, 16
  enddef;
  ndegree_values[2] = "25, 76, 100, 184, 196";
  ndegree_values[3] = "24, 65, 92, 171, 181";
  ...
  ndegree_values[16] = "131, 169, 182";
ndegree_min = 2; ndegree_max = 16;
def ndegree = scantokens ndegree_values enddef;
```

2.7 The edges

We have not dealt much with edges so far. Just as a node in MetaGraph is a number, a MetaGraph edge is a *pair* of numbers: the indices of the source and target nodes. As a graphical, coordinate-based language, METAPOST has a full infrastructure for tuples of 2 numbers — its data type `pair`. Edges are not coordinate pairs, of course, but they are METAPOST `pair`s.

The first implementation of MetaGraph treated the edges of a graph as an array of such pairs, each given an ⟨*edge-id*⟩ number. Edge attributes were analogous to node attributes: lists of ⟨*edge-id*⟩ numbers held in string variable arrays.

But the experiments with the Facebook graph in Figure 2 revealed something interesting, having to do with the assignment of an index number to each edge. That is: to each of all *88,234* of them... METAPOST users will know that this goes over the language's 4,096 arithmetic limit, but this is not the issue: there are ways to get around it (see section 3). The problem was that the program would take way, way too long — I could not bear to let it finish a single time — just reading the data file...

This seemed to doom the whole idea of MetaGraph, or at least limit it to relatively small graphs with a manageable number of edges. But it is easy to see that the '`edges`' array is redundant, especially as we have a list of neighbors for each node (see section 2.6). Storing the edges of the Facebook graph in this way entails at most 4,039 assignments (actually somewhat less, since not all nodes have outgoing neighbors). This is a significant reduction — enough to make the program run smoothly. The change makes it a little more complicated to loop over the edges, which now means looping over each node and then again over its `nodes_to`. But the nuisance is small enough, and the macro `alledges` implements the loop, even taking care of neighbor-less nodes (which would normally make the loop break).

Edge attributes are still possible, and they are still lists, analogous to those for nodes ("edges of attribute *foo* equal to value v are: these and these"), only holding `pair` values instead of single numbers. For example, this is the list of edges of frequency 2 in the Bach prelude graph:

```
efreq_values[2] = "(ii7, V7), (IVj2, ii7),
    (I6, IVj2), (V7, I)";
```

One can still loop over these lists and format the edges conditionally — the dark arrows in Figure 3 were ultimately a loop over this very list. Furthermore, nothing prevents one from creating new lists of edge attributes, or indeed new edges, as needed, in real time (i.e., outside the data file).

3 How much is too much?

Larger and larger graphs will of course exceed the system's capacity at some point. But even the 4,039 nodes and 88,234 edges of Figure 2 are far from exhausting METAPOST's (much less LuaTEX's) memory, stack sizes, etc. The graph, in fact, is processed by MetaGraph in a noticeably shorter time than matplotlib takes to draw it (and then again to display it, zoom over it, etc).

It is still a lucky coincidence that that particular graph has 4,039 nodes — dangerously close but still shy of METAPOST's arithmetic bound of 4,096. (This bound, you may recall, is not a matter of physical capacity, but a fundamental feature of the language: there simply exists no $n > 4096$ in METAPOST.)

This issue is not particularly hard to get around. The subscripts of a METAPOST array do not have to be natural numbers, and we could assign the attributes of a node of index i at position $\frac{i}{2}$ of the arrays — where in 'normal' circumstances we assign just i. This would give us up to 8,192 nodes. The same capacity would be achieved by wrapping around and using negative indices as well. Thus, indices of the form, say, $\pm i/10$, would give us 81,920 possible nodes.

Still, in an earlier draft of this paper, where the Facebook graph came after many other figures, the processing of the document *did* overflow Overleaf's free-version compile time. (The paid version, 12× faster, has no problem.) What can be done in those cases?

The solution is to produce a PNG through an external run of METAPOST, and `\includegraphics` it in the document. As a final example, the file that produced Figure 2 is reproduced below in its entirety. The first couple of lines are the ones that make METAPOST produce a PNG file ('`fbtopng-1.png`'); the rest is plain MetaGraph.

```
outputformat := "png";
outputtemplate := "%j-%c.png";

input metagraph;
input fbdata;

scale = 180;

beginfig(1);
  pickup pencircle scaled .1pt;
  for node = allnodes:
    for tgt = nodes_to[node]:
        draw pos[node] -- pos[tgt]
          withcolor .8white;
    endfor;
  endfor;

  addnodes;
endfig;
end.
```

◇ Federico García De Castro
Professor of Composition and Theory
EAFIT University
Medellín, Colombia
fgarciac1 (at) eafit dot edu dot co

Euclidean geometry with tkz-elements and tkz-euclide

Alain Matthes

Abstract

tkz-elements [2][1] is based on Lua and LuaLaTeX to perform calculations and obtain point coordinates in the plane. These coordinates are then transmitted to a package that can plot them. Currently, plotting is accomplished with TikZ or tkz-euclide, but MetaPost is also a viable option.

This paper demonstrates how tkz-elements can be utilized for tasks requiring mathematical computations. With it, not only can you create Euclidean geometry figures, but you can also conduct calculations within your document.

1 Introduction

The aim of the tkz-euclide [3] package is to provide a tool that would facilitate the construction of Euclidean geometric figures, with a key focus on being suitable for individuals who think mathematically, and even better, geometrically. tkz-euclide is built on top of PGF and its associated front-end TikZ. As a result, the calculations rely on TeX. To aid TeX in performing certain calculations, auxiliary packages are necessary. However, this approach can be challenging to program, slow in execution, and sometimes lacks accuracy.

An extension of TeX, LuaTeX, has been developed, enriching TeX with the programming language Lua, which is fast, light and easy to program. tkz-elements is an attempt to use Lua's capabilities to enhance tkz-euclide.

The final section of this paper explains the basics of drawing objects with tkz-euclide.

2 What are the foundations of tkz-elements?

2.1 Structure

The package mainly comprises two environments: the tkzelements environment and the tikzpicture environment. The former utilizes Lua-created functions to acquire point coordinates, while the latter employs tkz-euclide to draw figures. I have a preference for tkz-euclide, as it includes all the fundamental figures.

An important aspect is the relationship between the two environments. The coordinates of the points are stored in the only data structure available in Lua: a table z (z being a common reference to the affixes of complex numbers). This table is global, and its data is only cleared when a new tkzelements environment is initiated. At the start of the tikzpicture environment, the tkzGetNodes macro retrieves the coordinates and generates nodes whose names are those of the z table keys.[2]

Following the tkzelements environment, you can obtain results that can be incorporated into your document (an advantage of a figure source within your document), by using the \tkzUseLua command. The definition of this macro is
`\directlua{tex.print(tostring(#1))}`.

Let's look at the following example:[3]

```
% !TEX TS-program = lualatex
\documentclass{article}
\usepackage{tkz-euclide,tkz-elements}
\begin{document}
\begin{tkzelements} -- part elements
  z.A   = point : new (1,1)
  z.B   = point : new (3,2)
  C.AB  = circle : new (z.A,z.B)
  z.C   = C.AB : point (1/6)
  T.ABC = triangle : new (z.A,z.B,z.C)
\end{tkzelements}
\begin{tikzpicture}% part tikz
  \tkzGetNodes
  \tkzDrawPolygon(A,B,C)
  \tkzDrawCircle(A,B)
  \tkzDrawPoints(A,B,C)
  \tkzLabelPoints(A)
  \tkzLabelPoints[above](B,C)
\end{tikzpicture}

The length of AC is \tkzDN[4]{\tkzUseLua{%
                      length(z.C,z.A)}}
  Affix of $C$: \tkzUseLua{z.C}
\end{document}
```

Figure 1: Sample program

The result is shown in fig. 2.

The macro tkzDN serves as a formatting tool for numerical results.

Now, let's consider whether the triangle is equilateral. If the ifthen package has been loaded, this can be done with:

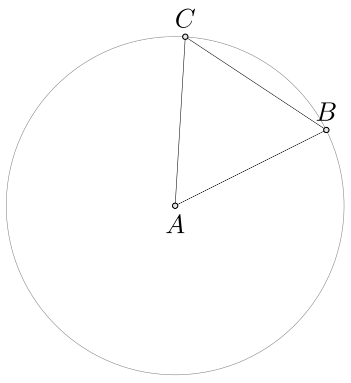
```
\ifthenelse{\equal{\tkzUseLua{%
  T.ABC : check_equilateral ()}}{true}}{%
  The triangle ABC is equilateral}{%
  The triangle ABC is not equilateral}
```

[1] The current version is 2.00 and is required to compile the examples in this paper.

[2] The table type implements associative arrays. An associative array is an array that can be indexed with numbers, strings, or any other value; that is, they store a set of key/value pairs.

[3] "`-- part elements`" is a comment in Lua; "`% part tikz`" is a comment in LaTeX.

doi.org/10.47397/tb/45-1/tb139matthes-euclid



The length of AC is 2.2361
Affix of C: 1.13+3.23i

Figure 2: Result of sample program fig-1

which, for our example, outputs:
The triangle ABC is equilateral

2.2 Tools

2.2.1 Complex numbers

Our primary aim was precision in calculations, and since programming with Lua is much easier than in TEX, we considered utilizing mathematical tools better suited to geometry instead of basic arithmetic operations like addition and subtraction. The initial concept was to incorporate complex numbers.

A complex number, denoted as z, can be represented by an ordered pair $(\mathcal{R}e(z), \mathcal{I}m(z))$ of real numbers, which can be interpreted as coordinates of a point in a two-dimensional space such as the Euclidean plane. This plane is commonly referred to as the complex plane or the Argand plane (Fig. 4). To create a point object, we specify its two coordinates and its name (future node name); for example: `z.A= point : new (2,3)`. What happens here? An object of type `point` is created, consisting of attributes and methods stored in the table (associative array) `z`.

The key `A` is associated with the data. The `tostring` method has been adapted to display the affix corresponding to the point. That is, `tex.print(tostring(z.A))` outputs `2+3i`.

Point objects behave similarly to the affixes that represent them. Hence, we can manipulate them with the same operations. Here's an example: adding two points means obtaining another point whose affix is the sum of the affixes of the previous points.

Let's consider a second point:
`z.B = point : new (2,-1)`
Then `z.C = z.A+z.B` has affix `4+2i`; analogously, `z.D = z.A*z.B` has affix `7+4i`.

Let's check: `\tkzUseLua{z.A*z.B}` computes: 7+4.00i.

Refer to the documentation for a comprehensive list of all methods available. Some are more significant than others, one being the complex conjugate: `z.B = z.A : conj()`, which can alternatively be expressed as `z.B = point.conj (z.A)`.

It's important to note that two operations have been repurposed from their conventional meanings: "`..`", typically represents concatenation but here denotes scalar product, and "`^`", usually signifies exponentiation but here denotes the determinant.[4]
`z.A .. z.B = (z.A * z.B : conj()).re` $= 1$
`z.A ^ z.B = (z.A : conj() * z.B).im` $= -8$

2.2.2 Barycenter

Another useful tool is the barycenter, which is utilized numerous times in our diagrams. Here are two examples demonstrating the advantages of combining complex numbers and barycenters:

- Obtaining the incenter in a triangle defined by its three vertices (a,b,c):

```
function in_center_ (a,b,c)
  local ka = point.abs (b-c)
  local kc = point.abs (b-a)
  local kb = point.abs (c-a)
  return barycenter_ ({a,ka},{b,kb},{c,kc})
end
```

  `point.abs` is a method which gives the modulus of a complex number.

- Obtaining the orthocenter:

```
function ortho_center_ (a,b,c)
  local ka = math.tan (get_angle_ (a,b,c))
  local kb = math.tan (get_angle_ (b,c,a))
  local kc = math.tan (get_angle_ (c,a,b))
  return barycenter_ ({a,ka},{b,kb},{c,kc})
end
```

  `get_angle_` is an internal macro in the package that produces a normalized angle defined by three complex numbers.

2.2.3 Objects — OOP

Finally, while the package's internal functions are classically programmed using Lua, user functions are based on object-oriented programming principles. Users manipulate points, lines, circles, triangles, etc., all of which are objects from specific classes. Currently, `tkz-elements` utilizes the following classes: point, line, circle, triangle, ellipse, quadrilateral, square, rectangle, parallelogram, regular (polygon) and vector (matrix will be added soon).

An object (or instance) of the class `point` has both state and behavior, defined by the class. The

[4] Here we consider `z.A` and `z.B` as the vectors $\overrightarrow{OA}$ and $\overrightarrow{OB}$ with O as the origin of the plane.

state is characterized by attributes, while behavior is determined by methods. The structure of the `object` class is shown in fig. 3; here, all the attributes are listed but only a few of the available methods are displayed. See [2], section “Class point” for the complete definition.[5]

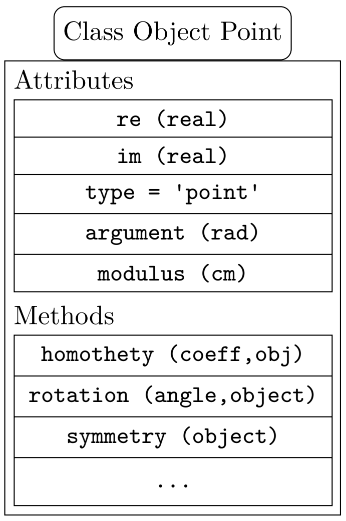


Figure 3: The Point object

We can access the instance’s attributes as follows to obtain the real part (the point’s abscissa): `z.A.re`.

We can already benefit from the use of LuaLaTeX. To obtain figure 4, the point A has been defined as follows `z.A = point : new (2,3)`. Therefore, we can use the attributes of this point. The modulus of z_A is 3.60555 . This value is obtained as follows: `\tkzUseLua{z.A.modulus}`

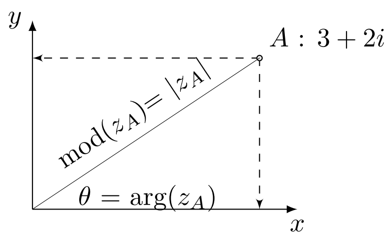


Figure 4: Argand diagram

Other classes possess their unique attributes and methods. We recommend consulting the documentation. In the remainder of this article, we’ll utilize examples to elucidate specific attributes and methods. It’s not feasible to cover all the documentation in this article, so we’ll employ examples to illustrate certain attributes and methods. Refer to [2], sections “Class line”, “Class circle”, etc.

[5] It’s recommended to have the package documentation at hand while reading this paper.

3 Small examples

Let’s examine two brief examples. While they don’t require high-precision calculations, they will demonstrate how to create a figure and utilize objects.

3.1 Alternate angles

```
\documentclass{article}
\usepackage{tkz-euclide}
\usepackage{tkz-elements}
\begin{document}
  \begin{tkzelements}
    scale = .8
     z.A   = point : new (0 , 0)
     z.B   = point : new (6 , 0)
     z.C   = point : new (1 , 5)
     T.ABC = triangle : new (z.A,z.B,z.C)
     L.AD  = T.ABC : bisector ()
     z.D   = L.AD.pb
     L.LLC = T.ABC.ab : ll_from (z.C)
     z.E   = intersection (L.AD,L.LLC)
  \end{tkzelements}
\begin{tikzpicture}
   \tkzGetNodes
   \tkzDrawPolygon(A,B,C)
   \tkzDrawLine(C,E)
   \tkzDrawSegment(A,E)
   \tkzMarkAngles[mark=|](B,A,D D,A,C)
   \tkzMarkAngles[mark=|](C,E,D)
   \tkzDrawPoints(A,...,E)
   \tkzLabelPoints(A,B)
   \tkzLabelPoints[above](C,D,E)
   \tkzMarkSegments[mark=s||](A,C C,E)
\end{tikzpicture}
\end{document}
```

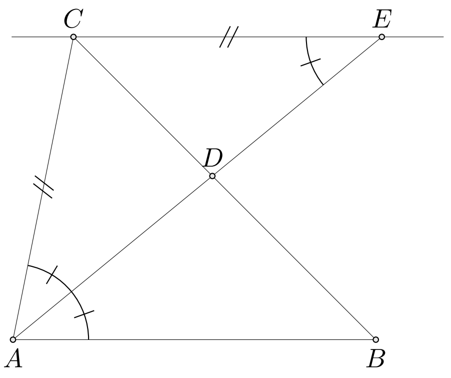


Figure 5: Alternate angles

First, we create three points, then a triangle named `T.ABC`. Subsequently, we define the bisector emanating from vertex A.

`L.AD = T.ABC : bisector ()`: The bisector is defined by two points: the vertex A and the foot D on the opposite side. For the bisector from B you need to use `L.BE = T.ABC : bisector (1)`, where 1 is for the next point of the triangle.

To find the intersection of the bisector with a line parallel to the (AB) line at C, we'd have to name this line, but this is already done in the triangle's attributes: `T.ABC.ab` represents the triangle line defined by the first and second vertices. Finally, `z.E = intersection (L.AD,L.LLC)` gives the last point.

The `tkz-euclide` section gives an overview of the possibilities the package provides to mark segments and angles.

3.2 An Apollonius circle

Given k a positive real number other than 1, and A and B two points in the plane, the set of points M verifying $MA/MB = k$ is a so-called Apollonius circle. In the example below, k is defined by $k = EA/EB$.

```
\begin{tkzelements}
  scale = .5
  z.A   = point : new (0,0)
  z.B   = point : new (6.5,0)
  z.E   = point : new (7,4)
  T.EAB = triangle : new (z.E,z.A,z.B)
  EA    = length (z.E,z.A)
  EB    = length (z.E,z.B)
  C.OE  = T.EAB.bc : apollonius (EA/EB)
  L.bis = T.EAB : bisector ()
  z.C   = L.bis.pb
  z.O   = C.OE.center
  z.D   = C.OE : antipode (z.C)
  z.F   = T.EAB.ab : point (-0.5)
\end{tkzelements}
```

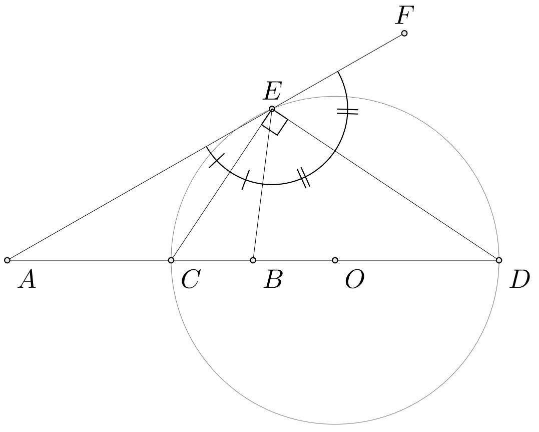


Figure 6: Apollonius $MA/MB = k$

- We define the triangle after defining three points:
  `T.EAB = triangle : new (z.E,z.A,z.B)`
- The length EA is determined with
  `length(z.E,z.A)`
- `T.EAB.bc` represents the straight line (AB) b for the second point and c for the third. Find the circle defined by these two points and the ratio EA/EB. It is called `C.EC` because its center will be O and it passes through E.
- We get the circle with
  `T.EAB.bc : apollonius (EA/EB)`
- Next, we look for the bisector of the angle $\widehat{AEB}$. It intersects the opposite side at point C of $C_{(O,E)}$.

  In `T.EAB : bisector ()`, the first point designates the vertex. The bisector is defined by the vertex and the intersection with the opposite side; `L.bis.pb` designates the second point.
- In `z.O = C.OE.center`, `center` is a circle attribute, then the "antipode" method is used to obtain the diametrically opposite point
  `z.D   = C.OE : antipode (z.C)`.
- Finally we need a point F to mark an angle in `tkz-euclide`.

4 Harmonic mean of two numbers

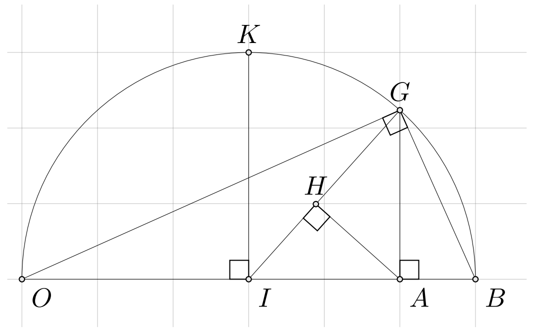


Figure 7: Means of two numbers

For two numbers a and b, such as $OA = a$ and $AB = b$, here are the definitions and geometric representations of three means:

Mean	Definition	Segment
Arithmetic	$\mathcal{A} = \dfrac{a+b}{2}$	IK
Geometric	$\mathcal{G} = \sqrt{ab}$	AG
Harmonic	$\mathcal{H} = \dfrac{2ab}{a+b} = \dfrac{\mathcal{G}^2}{\mathcal{A}}$	HG

```
\begin{tkzelements}
  local a = 5
  local b = 1
  z.O    = point : new (0,0)
  z.A    = point : new (a,0)
  z.B    = z.A + b
  L.OB   = line : new (z.O,z.B)
  z.I    = L.OB.mid
  C.IO   = circle : new (z.I,z.O)
  L.orth = L.OB : ortho_from (z.A)
  z.K    = C.IO.north
  z.G,z.Gp = intersection (L.orth,C.IO)
  L.IG   = line : new (z.I,z.G)
  z.H    = L.IG : projection (z.A)
\end{tkzelements}
```

Tracing with `tkz-euclide`:

```
\begin{tikzpicture}[gridded]
  \tkzGetNodes
  \tkzDrawSegments(I,G A,H O,B)
  \tkzDrawSegments(O,G G,B I,K A,G)
  \tkzDrawArc(I,B)(O)
  \tkzLabelPoints[below right](O,A,B,I)
  \tkzLabelPoints[above](H,K,G)
  \tkzMarkRightAngles(O,I,K B,A,G)
  \tkzMarkRightAngles(A,H,I O,G,B)
  \tkzDrawPoints(O,A,B,G,K,H,I)
\end{tikzpicture}
```

Some explanations:

- `z.B = z.A + b`
  Adding points means adding their corresponding affixes. `z.A` is represented in this equation by the affix, so it's possible to add a real or complex number to it. We have $OB = a + b$.
- `L.OB = line : new  z.O,z.B)`: create a `line` object with key `OB`. Then, in
  `z.I = L.OB.mid`, `mid` is an attribute of the `line` object giving the midpoint of the segment defined by the two points characterizing the line.
- `C.IO = circle : new (z.I,z.O)`: create a `circle` object with key `IO`.
- In `z.K = C.IO.north`, the `north` attribute of a circle is used.
- This is followed by an intersection:
  `intersection (L.orth,C.IO)`
  The arguments are objects, given in no particular order. Depending on the object types, the function selects the correct algorithm.
  The two points of intersection will be G and G' (`Gp` in Lua for the moment).
- `projection` is a method of the `line` object.

Let's check some data:

- The coordinates of G are (5 ; 2.2361) with
  `\tkzUseLua{z.G.re}` ; `\tkzUseLua{z.G.im}`
- The coordinates of H are (3.8889 ; 0.9938)
- The harmonic mean is the length of GH = 2.2361 , i.e., $\sqrt{5}$ with
  `\tkzUseLua{length(z.G,z.A)}`

```
\begin{tkzelements}
  scale =.8
  dofile ("means_b.lua")
\end{tkzelements}
```

It's good practice to place the Lua code in an external file. This approach makes it easier to correct and reuse, and it helps avoid errors when using special characters like the % symbol.

Figure 9 illustrates how to obtain half the harmonic mean and, importantly, demonstrates that this method is independent of the distance d.

```
z.A   = point : new (0,6)
z.B   = point : new (6,4)
z.Bp  = point : new (8,4)
z.I   = point : new (0,0)
z.J   = point : new (6,0)
z.Jp  = point : new (8,0)
L.AJ  = line : new (z.A,z.J)
L.IJ  = line : new (z.I,z.J)
L.BI  = line : new (z.B,z.I)
z.C   = intersection (L.AJ,L.BI)
z.K   = L.IJ : projection (z.C)
L.AJp = line : new (z.A,z.Jp)
L.BpI = line : new (z.Bp,z.I)
z.Cp  = intersection (L.AJp,L.BpI)
z.Kp  = L.IJ : projection (z.Cp)
```

Figure 8: File `means_b.lua`

```
\begin{tikzpicture}
  \tkzSetUpPoint[size=8]
  \tkzGetNodes
  \tkzDrawSegments[dashed](A,J B,I I,J)
  \tkzDrawSegments[dashed](A,J' B',I)
  \tkzDrawPoints[gray,size = 8](A,I,C,K,B,J)
  \tkzDrawPoints[black,size = 8](C',K',B',J')
  \tkzSetUpLine[ultra thick]
  \tkzDrawSegments[black](C',K' B',J')
  \tkzDrawSegments[gray](C,K A,I B,J)
\end{tikzpicture}
```

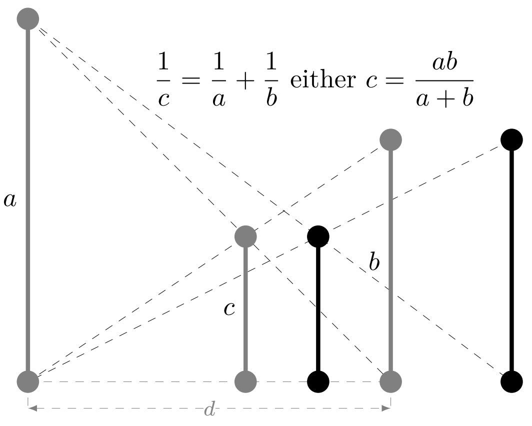

$\frac{1}{c} = \frac{1}{a} + \frac{1}{b}$ either $c = \frac{ab}{a+b}$
a
b
c
d

Figure 9: Half of harmonic mean

5 THE Apollonius circle

The circle that touches all three excircles of a triangle and encompasses them is commonly referred to as

"THE" Apollonius circle. Our approach is from the fourth definition given in [4], due to Kimberling [1, p. 102].

The objective here is to determine the external tangent circle to the three exinscribed circles of a triangle. While this problem is mathematically challenging, the idea is to demonstrate that the package offers some highly useful capabilities for experienced geometers.

The approach involves determining the inner tangent circle, and then transforming this inner circle into an outer circle, also tangent to the exinscribed circles. The result is shown in fig. 10.

The Lua code is created in an external file, `apollonius.lua`, shown in fig. 11.

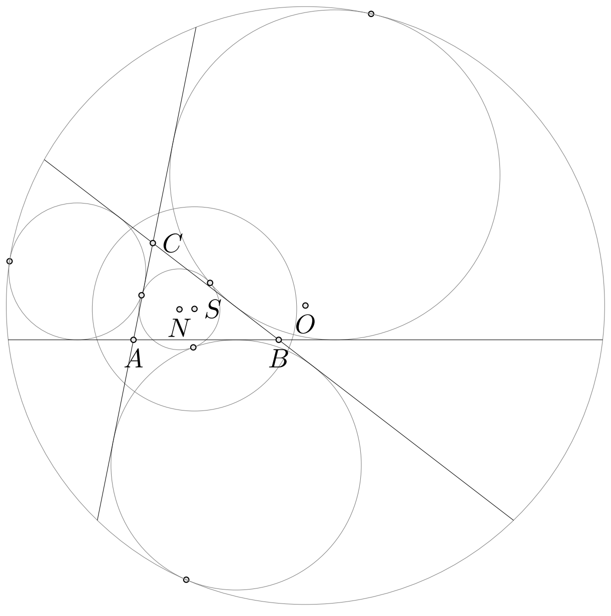


Figure 10: THE Apollonius circle

5.1 Code analysis

- A `triangle` object is created: `T.ABC`, then we utilize its attributes and methods linked to the `triangle` class.
- For example, `z.N` refers to the Euler center or the center of the nine-point circle. Additionally, `T.feuerbach` is a triangle created using a method. Its vertices are the points of contact of the Euler circle with the exinscribed circles.
- Then, to draw them, we'll need the points that define the vertices of `T.feuerbach`. This is the role of `get_points (T.feuerbach)`.

  `get_points` is a function that retrieves the points (attributes) required to create the object. In this case, these are the vertices of the triangle

```
scale       = .32
z.A         = point : new (0,0)
z.B         = point : new (6,0)
z.C         = point : new (0.8,4)
T.ABC       = triangle : new (z.A,z.B,z.C)
z.N         = T.ABC.eulercenter
T.feuerbach = T.ABC : feuerbach ()
T.excentral = T.ABC : excentral ()
z.Ea,z.Eb,z.Ec = get_points (T.feuerbach)
z.Ja,z.Jb,z.Jc = get_points (T.excentral)
z.S         = T.ABC.spiekercenter
C.JaEa      = circle : new (z.Ja,z.Ea)
r_ortho     = math.sqrt (C.JaEa : power (z.S))
C.ortho     = circle : radius (z.S,r_ortho)
z.a         = C.ortho.south
C.euler     = T.ABC : euler_circle ()
C.apo       = C.ortho : inversion (C.euler)
z.O         = C.apo.center
z.xa,
z.xb,
z.xc = C.ortho : inversion (z.Ea,z.Eb,z.Ec)
```

Figure 11: File `apollonius.lua`

Ea, Eb, Ec. The circle with center N passes through these points.

- The same procedure is used to recover the centers of the exinscribed circles (Ja, Jb, Jc).
- On a more technical note, the radical axes of the three exinscribed circles intersect at a point called the "radical center", which is none other than the Spieker center. This point is known to the package as one of the attributes of the triangle: `z.S = T.ABC.spiekercenter`.

  It's also possible to directly request the radical center. The radical center has the same power with respect to the three circles. This allows for determining the radius of a circle that will be orthogonal to the three exinscribed circles. The radius is
  `r_ortho = math.sqrt (C.JaEa : power (z.S))`.

  Calculate the power of point S with respect to one of the three circles, then take the square root of the result.
- The circle "ortho" can be defined as
  `C.ortho = circle : radius(z.S,r_ortho)`.

  All that remains is to utilize this circle to perform an inversion of the Euler circle, which will give the Apollonius circle
  `C.apo = C.ortho : inversion(C.euler)`.

  We then retrieve the center
  `z.O = C.apo.center` (center is an attribute for a circle) and the three points of contact with the exinscribed circles. These are images of the inverted contact points of the nine-point circle or Euler circle.

```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawPoints(xa,xb,xc)
  \tkzDrawCircles(Ja,Ea Jb,Eb Jc,Ec S,a O,xa N,Ea)
  \tkzClipCircle(O,xa)
  \tkzDrawLines[add=3 and 3](A,B A,C B,C)
  \tkzDrawPoints(O,A,B,C,S,Ea,Eb,Ec,N)
  \tkzLabelPoints(O,N,A,B)
  \tkzLabelPoints[right](S,C)
\end{tikzpicture}
```

6 Kissing circles

6.1 The problem

Given three circles tangent to each other and to a straight line, the problem is to express the radius of the middle circle in terms of the radii of the other two. This problem was presented as a Japanese temple problem on a tablet from 1824 in the Gunma Prefecture (MathWorld). [5]

While not overly complicated, the construction and justification with ruler and compass are interesting. The desired output is shown in fig. 12.

The first step is to create a function to obtain the centers of the three circles, and then to determine the projections of these centers onto the common tangent of the three circles.

We call the function responsible for doing this `kissing` (fig. 13). In the following example, A, B and C represent the centers of the circles, 4 and 3 the radii of the two given circles, and E, F and G the projections of the centers.

Additionally, the function defines several useful objects such as straight lines `L.AB`, `L.EF`, and circles `C.AE`, `C.BF` and `C.CH`.

It's worth noting that the function uses the normal syntax `L[c1..c2]` instead of the "syntactic sugar" `L.name`. While the function's logic is not overly complex, attention to syntax is essential for proper execution.

```
\begin{tkzelements}
  dofile ("kissing.lua")
\end{tkzelements}
```

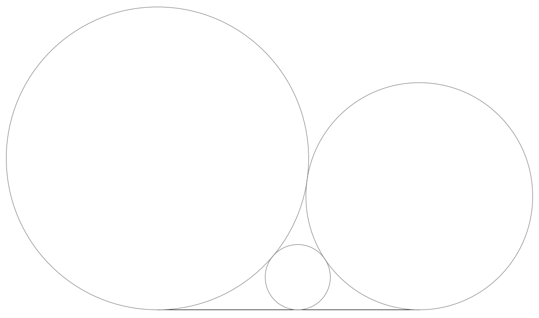


Figure 12: Three tangent circles

```
  function kissing(c1,r1,c2,r2,c3,h1,h3,h2)
    local xk  = math.sqrt (r1*r2)
    local de  = math.sqrt (r1) + math.sqrt (r2)
    local cx  = (2*r1*math.sqrt(r2))/de
    local cy  = (r1*r2)/(de^2)
    z[c2]     = point : new (2*xk,r2)
    z[h2]     = point : new (2*xk,0)
    z[c1]     = point : new (0,r1)
    z[h1]     = point : new (0,0)
    L[c1..c2] = line : new (z[c1],z[c2])
    L[h1..h2] = line : new (z[h1],z[h2])
    z[c3]     = point : new (cx,cy)
    z[h3]     = L[h1..h2] : projection (z[c3])
    C[c1..h1] = circle : new (z[c1],z[h1])
    C[c2..h2] = circle : new (z[c2],z[h2])
    C[c3..h3] = circle : new (z[c3],z[h3])
  end
```

Figure 13: The "kissing" function code

```
\begin{tkzelements}
  scale = .5
  kissing ("A",4,"B",3,"C","E","G","F")
  L.AE  = line : new (z.A,z.E)
  z.H   = L.AE : projection (z.B)
\end{tkzelements}
```

Now the code for the TikZ part:

```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegment(E,F)
  \tkzDrawCircles(A,E B,F C,G)
\end{tikzpicture}
```

6.2 Construction with an inversion

The diagram for a construction with an inversion is shown in fig. 14.

```
\begin{tkzelements}
   scale    = .92
   dofile ("kissing.lua")
   kissing ("A",4,"B",2,"C","E","G","F")
   z.X      = intersection (C.AE,C.CG)
   z.Y      = intersection (C.BF,C.CG)
   z.T      = intersection (L.AB,C.AE)
   z.H      = L.EF : projection (z.T)
   z.O      = midpoint (z.T,z.H)
   C.TH     = circle : new (z.T,z.H)
   z.x,z.xp = intersection (C.AE,C.TH)
   z.y,z.yp = intersection (C.BF,C.TH)
   z.x,z.xp = intersection (C.AE,C.TH)
   if z.x.re < z.xp.re then else
     z.x,z.xp = swap (z.x,z.xp) end
   z.y,z.yp = intersection (C.BF,C.TH)
   if z.y.re < z.yp.re then else
     z.y,z.yp = swap (z.y,z.yp) end
   L.OS     = L.AB : ortho_from (z.O)
   C.O      = circle : new (z.O,z.H)
   _,z.S    = intersection (L.OS,C.O)
   z.W      = z.S : symmetry (z.O)
   z.Np     = z.W : symmetry (z.S)
   z.Ep,z.Fp,
   z.N      = C.TH : inversion (z.E,z.F,z.Np)
   z.Xp,z.Yp= C.TH : inversion (z.X,z.Y)
   T.EFN    = triangle : new (z.E,z.F,z.N)
   T.EFNp   = triangle : new (z.E,z.F,z.Np)
   z.I      = T.EFN .circumcenter
   z.Ip     = T.EFNp.circumcenter
   z.Bn     = C.BF.north
   z.Fp     = z.Bn : symmetry (z.F)
\end{tkzelements}
```

6.2.1 Lua code analysis

After calling `kissing`, several points are defined such as A, B, …, G. Additionally, circles `C.AE`, `C.BF`, `C.CG` and lines `L.AB` and `L.EF` are defined.

- We designate as X, Y and T the contact points between the three circles. These points are obtained through intersections, for example: `z.X = intersection (C.AE,C.CG)`.
- H is obtained by projecting T onto the line (EF): `z.H = L.EF : projection (z.T)`.
- To maintain consistent notation, a test is conducted to ensure that x and y are closest to the line (EF). Depending on the results, the points x, x' and y, y' may be exchanged.
- `L.OS = L.AB : ortho_from (z.O)` is defined as the orthogonal line to (AB) passing through O.
- The method `symmetry` attached to points is utilized to determine point W, which is the symmetric of O with respect to S. This is obtained with: `z.W = z.S : symmetry (z.O)`.
- Finally, points E', F' and N are obtained by inversion with respect to the circle with center T passing through H. This circle is denoted `C.TH` and the transformations of the points are obtained with:
  `z.Ep,z.Fp,`
  `z.N = C.TH : inversion (z.E,z.F,z.Np)`.

  Note the use of the letter `p` in the point names, which indicates the “prime” when converting points to nodes.
- The remaining steps involve using attributes and methods which we’ve already discussed.

```
\begin{tikzpicture}
  \tkzGetNodes
  \tkzDrawSegments(E,F A,B E,A F,B A,C Bn,E)
  \tkzDrawSegments[lightgray](T,X' T,Y' T,N')
  \tkzDrawCircles(B,F T,H)
  \tkzDrawCircles[](C,G)
  \tkzDrawCircle[](O,H)
  \tkzDrawCircle[](W,S)
  \tkzDrawArc[delta=10](A,E)(x')
```

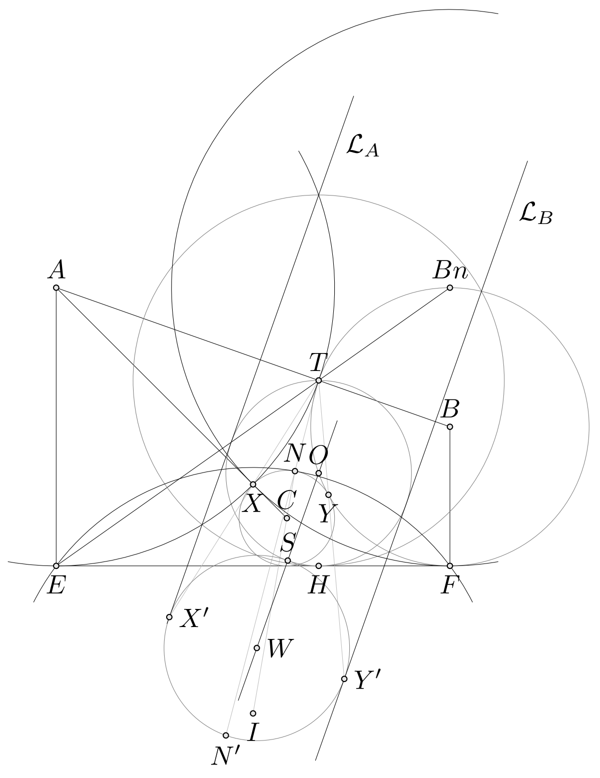


Figure 14: Method with inversion

```
  \tkzDrawArc[delta=10](I,F)(E)
  \tkzDrawArc[delta=10](Bn,F')(F)
  \tkzDrawLines[add=.3 and 0.3](x,x' O,W)
  \tkzDrawLines[add=.8 and 0.5](y,y')
  \tkzDrawPoints(A,B,E,F,T,S,W,C,H,X,Y)
   \tkzDrawPoints(X',Y',N',N,Bn,O)
  \tkzLabelPoints(E,F,H,X,Y,N')
  \tkzLabelPoints[right](X',Y',W)
  \tkzLabelPoints[above](S,Bn,N,A,B,O,T,C)
  \tkzLabelLine[pos = 1.15,right]%
      (x,x'){$\mathcal{L}_A$}
  \tkzLabelLine[pos = 1.3,right]%
      (y,y'){$\mathcal{L}_B$}
\end{tikzpicture}
```

7 Drawing with tkz-euclide

If you’re utilizing `tkz-elements` and intend to use Ti*k*Z, the macro `\tkzGetNodes` is essential. It generates nodes from points defined in the `tkzelements` environment.

7.1 A few basics

1. Drawing: The role of `tkz-euclide` is minimized in drawing simple Euclidean geometry objects.

- Points: `\tkzDrawPoints(A,B,C)`
- Segments: `\tkzDrawSegements(A,B C,D)`
- Lines: `\tkzDrawLines(A,B C,D)`

- Circles: `\tkzDrawCircles(A,B C,D)`[6]
- Polygons: `\tkzDrawPolygons(A,B,C D,E,F)`[7]
- Ellipse: `\tkzDrawLuaEllipse(C,A,B)`[8]

  You can define the styles of objects globally or use a style locally. For example:
  `\tkzDrawPoints[style](A,B,C)`.

2. Marking: Additionally, you have the option to mark segments or angles.
   - `\tkzMarkSegments[s|](A,B C,D)`
   - `\tkzMarkArc(O,A)(B)`
   - `\tkzMarkAngles(A,B,C)`
   - `\tlkzMarkRightAngles(A,B,C)`
3. Labeling:
   - `\tkzLabelPoints(A,B,C)`
   - `\tkzLabelSegments(A,B C,D)`
   - `\tkzLabelAngle(A,B,C){$\alpha$}`
   - `\tkzLabelCircle(O,A)(60){$C(O,A)$}`

7.2 Styling

The `tkz-euclide` package includes a configuration file `tkz-euclide.cfg` containing all style definitions, which can be duplicated and modified as needed. Let's explore the methods for changing point styles; the principle will be identical for other objects.

7.2.1 Styling the points

Points: To draw points A, B and C, you can use `\tkzDrawPoints(A,B,C)`. This is the same as Ti*k*Z. In `tkz-euclide`, points are represented as Ti*k*Z `coordinates`.

Here are some additional details on styling points in `tkz-euclide`:

- Setting global point size: You can set the global point size for the entire figure or document.
  `\tkzSetUpPoint[size=.8pt]`

  You can also change this size locally when needed. In some cases, you may need to use a group or a scope for local modification.
- Creating local styles: You can create local styles by customizing the style name. For example:
  `\tikzset{step 1/.style={cyan,thin}}` and
  `\tikzset{step 2/.style={red,thick}}`
  which you can use in this way:
  `\tkzDrawPoints[step 1](A,B)` and
  `\tkzDrawPoints[step 2](C)`
- Combining general and specific styles: You can define a general style and then create adaptations from it. For example:
  `\tkzSetUpPoint[size=.8pt]`

[6] center A through B
[7] triangle ABC
[8] C = center, A = vertex, B = covertex

- Modifying predefined styles: It's possible to modify predefined styles directly:
  `\tikzset{point style/.style={...}}`
- Retaining and modifying predefined styles: You can retain part of a predefined style and add to or modify it as needed.
  `\tikzset{point style/.append style={}}`
- Finally, you can create your own local style from a global style as follows:

  ```
  \tikzset{new/.style={point style/
  .append style={minimum size=8 pt,
  fill=green}}}
  ```

  This allows you to build upon a global style and make specific modifications for local use.

7.2.2 Styling other objects

Besides `point style`, you can look at, modify, etc., these other styles:

- `line style`
- `circle style`
- `compass style`
- `arc style`
- `vector style`

References

[1] C. Kimberling. Triangle centers and central triangles. *Congressus Numerantium*, 129:1–295, 1998.

[2] A. Matthes. tkz-elements 2.00c, 2024. `ctan.org/pkg/tkz-elements`

[3] A. Matthes. tkz-euclide 5.06c, 2024. `ctan.org/pkg/tkz-euclide`

[4] E.W. Weisstein. Apollonius circle. From MathWorld — A Wolfram Web Resource, n.d. `mathworld.wolfram.com/ApolloniusCircle.html`

[5] E.W. Weisstein. Tangent circles. From MathWorld — A Wolfram Web Resource, n.d. `mathworld.wolfram.com/TangentCircles.html`

◇ Alain Matthes
5 rue de Valence
Paris V, 75005
France
`alain (dot) matthes (at) mac (dot) com`
`https://altermundus.fr`

Unusual bitmaps

Hans Hagen, Mikael P. Sundqvist

1 Introduction

In the early days of TeX, fonts mostly were bitmaps and when such a bitmap was shown in zeros and ones, the shape was rather recognizable. A recent example of a special-purpose (TAOCP) bitmap font is Don Knuths three–six font. Do you recognize this character?

```
1111111111
1100110011
1100110011
0000110000
0000110000
0000110000
0001111000
0001111000
```

It has a rather low resolution, but can still serve its purpose as we will see later on. One problem is of course that such a low resolution doesn't render too well. Although vector images are often preferred, especially in a MetaPost context, below we will explain how we can also use bitmaps in a constructive way.

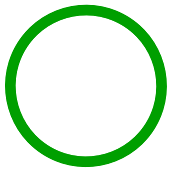
2 Vectorizing bitmaps

The potrace library by Peter Selinger is a nice tool: you feed it a bitmap and are rewarded with an outline specification. Details about the process can be found in `https://potrace.sourceforge.net/potrace.pdf`. When you limit yourself to only the basics, there are not that many source files and therefore I decided to add it to LuaMetaTeX in order to explore if we can do runtime conversions. Possible applications are logos and maybe converted bitmap fonts, although these can best be prepared beforehand because consistent metrics need to be taken care of. There are, however, other applications possible. Here I will discuss usage only in the perspective of Metafun, simply because we have to be visual, and Metafun is all about that. The library is of course accessible from the Lua end, if only because that way we can use it in Metafun.

We start with a simple example that also shows a potential usage:

```
\startMPcode
    string s ; s := "010 111 010";
    draw lmt_potraced [
        bytes = s,
    ] ysized 2cm
      withpen pencircle scaled 4
      withcolor darkgreen ;
\stopMPcode
```

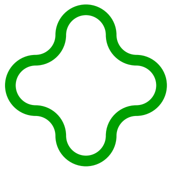
We feed the `lmt_potraced` macro a 3×3 bitmap encoded as a string and this is what we get back:



We expect a cross and get back a circle which is not what we want. This is because we don't have much body in this bitmap and potrace needs more pixels in order to give back a decent outline.

```
\startMPcode
    string s ; s := "010 111 010";
    draw lmt_potraced [
        bytes   = s,
        explode = true,
    ] ysized 2cm
      withpen pencircle scaled 4
      withcolor darkgreen ;
\stopMPcode
```

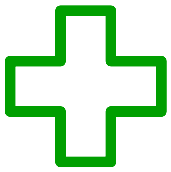
So, this time we 'explode' the bitmap, that is: we repeat every pixel three times in the horizontal and vertical direction. One can specify `nx` and `ny` but so far using different values doesn't help more than the magic threesome.



We're getting there but need to do a bit more.

```
\startMPcode
    string s ; s := "010 111 010";
    draw lmt_potraced [
        bytes     = s,
        threshold = 0.25,
        explode   = true,
    ] ysized 2cm
      withpen pencircle scaled 4
      withcolor darkgreen ;
\stopMPcode
```

Here we've set a threshold which will clip the paths in a range so that our case will get a better fit:

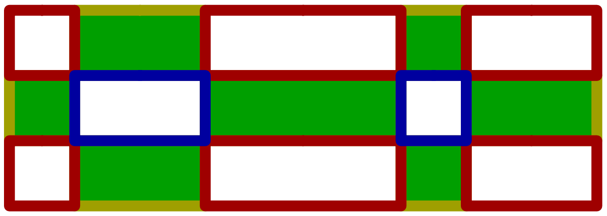


By now you will have noticed that we get back a path and this is an important feature of potrace: it returns a closed path expressed in lines and curves that one is supposed to fill. That result is converted into a Lua table that we then can use for instance to generate a valid MetaPost path. We can do whatever we like with that path: draw or fill it for clipping. Natively, the library might return multiple paths but the user sees only one because we concatenate them which is a feature of the MetaPost library that comes with LuaMetaTEX.

Once we could do this it was no big deal to add support for filtering. After all, we have more than `0` and `1` characters available. Take this example, where we lay out the bitmap a bit differently, for clarity:

```
\startMPcode
  string s ; s := "
      211222122
      133111311
      211222122
  ";
  path p[] ;
  p[1] := lmt_potraced [
      bytes     = s,
      threshold = 0.25,
      explode   = true,
      value     = "1",
  ] ;
  p[2] := lmt_potraced [
      bytes     = s,
      threshold = 0.25,
      explode   = true,
      value     = "2",
  ] ;
  p[3] := lmt_potraced [
      bytes     = s,
      threshold = 0.25,
      explode   = true,
      value     = "3",
  ] ;

  fill p[1] withcolor darkgreen ;
  draw p[1] withcolor darkyellow ;
  draw p[2] withcolor darkred ;
  draw p[3] withcolor darkblue ;

  currentpicture
    := currentpicture xsized TextWidth ;
\stopMPcode
```

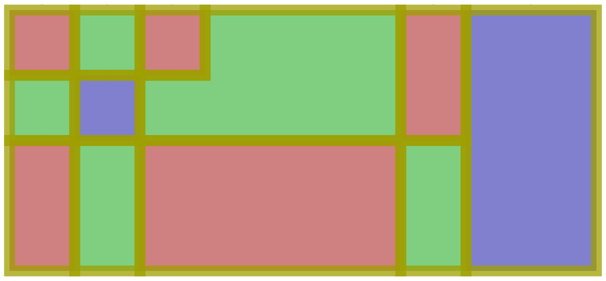
We save the paths so that we can use them multiple times, here for a draw and fill operation on the first path but you could scale, rotate, or manipulate the result before rendering it.



This example demonstrates that a user can define outlines using a bitmap specification and that the amount of code is rather small. At some point we might add a few more helpers that might reduce the amount of code even more.

```
\startMPcode
    string s ; s := "
        212111233
        131111233
        212222133
        212222133   ";
    path p[] ;
    p[1] := lmt_potraced [
        bytes     = s,
        threshold = 0.25,
        explode   = true, value = "1",
    ] ;
    p[2] := lmt_potraced [
        bytes     = s,
        threshold = 0.25,
        explode   = true, value = "2",
    ] ;
    p[3] := lmt_potraced [
        bytes     = s,
        threshold = 0.25,
        explode   = true, value = "3",
    ] ;
    linejoin := butt ;
    fill p[1] withcolor darkgreen
              withtransparency (1,.50) ;
    fill p[2] withcolor darkred
              withtransparency (1,.50) ;
    fill p[3] withcolor darkblue
              withtransparency (1,.50) ;
    draw p[1] withcolor darkyellow
              withtransparency (1,.75) ;
    draw p[2] withcolor darkyellow
              withtransparency (1,.75) ;
    draw p[3] withcolor darkyellow
              withtransparency (1,.75) ;
  currentpicture :=
    := currentpicture xsized TextWidth ;
\stopMPcode
```

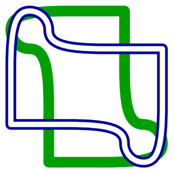
Because we have single paths we can safely apply properties like transparency, as shown below: crossing lines come out right instead of with accumulated transparent colors.



Sometimes you want to swap the rows and columns so we provide a feature for doing that:

```
\startMPcode
    string s[] ;

    s[1] := "1110 0110 0110 0111";
    s[2] := "1000 1111 1111 0001";

    draw lmt_potraced [
        bytes   = s[1],
        explode = true,
    ] ysized 2cm withcolor darkgreen
                 withpen pencircle scaled 4 ;

    draw lmt_potraced [
        bytes   = s[2],
        explode = true,
    ] ysized 2cm withcolor darkblue
                 withpen pencircle scaled 4 ;

    draw lmt_potraced [
        bytes   = s[1],
        explode = true,
        swap    = true,
    ] ysized 2cm withcolor white
                 withpen pencircle scaled 2 ;
\stopMPcode
```

When one uses Lua input (as we will see later), one can do that when generating the bitmap. At any rate, this is what we get from the above:



The previous examples demonstrate that not much code is needed in order to achieve nice effects. It also illustrates that one needs to twist the mind a little and think of bitmap specifications as actually efficient outline definitions. One could argue that such rectangular shapes are easy to program in MetaPost anyway, and going via bitmaps is kind of strange. So, let's move on to a more attractive example.

3 How about fonts

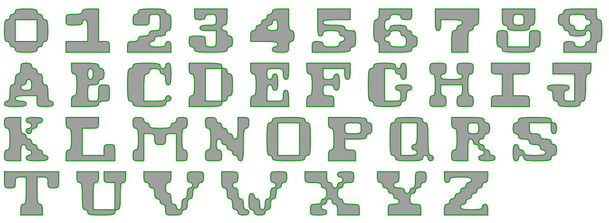
In order to demonstrate building fonts we need a decent bitmap font and it happens that Don Knuth's 'Font36' is a good candidate (`https://erikdemaine.org/fonts/dissect`). We have discussed that one elsewhere and its usage can be found in the Metafun `threesix` library. We happily borrow the definitions from that library (actual source strings are all on one line):

```
\startMPdefinitions
string dekthreesix[] ; path shapes[] ;

def DEK(expr n, b) =
    dekthreesix[utfnum(n)] := b ;
enddef ;

DEK("0", "00111100 01111110 11000011 11000011
          11000011 11000011 01111110 00111100");
...
DEK("Z", "11111111 10000111 00001110 00011100
          00111000 01110000 11100001 11111111");
\stopMPdefinitions
```

We use these definitions in the MetaPost code below. Helpers like `utfnum` are part of LuaMetafun and we use (named) colors defined at the ConTEXt end.

```
\startMPcode
def shapethem(expr first, last) =
    for i = utfnum(first) upto utfnum(last) :
        shapes[i] := lmt_potraced [
            bytes   = dekthreesix[i],
            explode = true,
          % threshold = .25, % more rectangular
            value   = "1",
        ] ;
    endfor ;
enddef ;

def drawthem(expr first, last, dx, dy) =
    numeric d ; d := 0 ;
    numeric f ; f := utfnum(first) ;
    numeric l ; l := utfnum(last) ;
    for i = f upto l :
        fill shapes[i] shifted (d,dy)
            withcolor "middlegray" ;
        draw shapes[i] shifted (d,dy)
             withcolor "darkgreen" ;
      % draw boundingbox shapes[i] shifted (d,dy);
        d := d + bbwidth(shapes[i]) + dx;
    endfor ;
enddef ;

shapethem("0","9") ;
shapethem("A","Z") ;

drawthem("0", "9", 10, -00) ;
drawthem("A", "J", 10, -30) ;
drawthem("K", "S", 10, -60) ;
drawthem("T", "Z", 10, -90) ;

currentpicture
  := currentpicture xsized TextWidth ;
\stopMPcode
```

Here we don't integrate it as a font but just show the characters as they come out, which hopefully is easier to understand. You can take a close look at the bitmaps in order to see what we get rendered below. Setting a lower threshold will give more rectangular results.



Because we're not going to use this font for typesetting here, a simple example of a definition will do. We first load an already existing module so that we don't need to define the bitmap definitions; basically they are the same as above.

```
\useMPlibrary[threesix]

\startMPcalculation{simplefun}
    vardef ThreeSixPotraced
                  (expr code, spread, lift) =
        draw lmt_potraced [
            bytes   = code,
            explode = true,
            value   = "1",
        ] scaled (1/3) shifted (spread,lift) ;
    enddef ;
\stopMPcalculation
```

Next we register a Metafun font using similar trickery as in that module. There we define a few variants but that is not needed here.

```
\startluacode
  local utfbyte = utf.byte
  local f_code = string.formatters
           ['ThreeSixPotraced("%s",%s,%s);']

  function MP.registerthreesixpotraced(name)
    fonts.dropins.registerglyphs {
        name     = name,
        units    = 12,
        usecolor = true,
    }
    for u, data in table.sortedhash(MP.font36) do
        local ny     = 8
        local nx     = ((#data + 1) // ny) - 1
        local height = ny * 1.1 - 0.1
        local width  = nx * 1.1 - 0.1
        local spread = 0.9
        local lift   = 0.3
        fonts.dropins.registerglyph {
            category = name,
            unicode  = utfbyte(u),
            width    = width + spread,
            height   = height,
            code     = f_code(data,spread,lift),
        }
    end
  end
  MP.registerthreesixpotraced
    ("fontthreesixpotraced")
\stopluacode
```

Finally we define a font feature that will hook the previous code into the font handler. There a Type 3 font will be constructed.

```
\definefontfeature
  [fontthreesixpotraced]
  [default]
  [metapost=fontthreesixpotraced,
   spacing=.375 plus .2 minus .1 extra .375]

\definefont[DEKFontP][Serif*fontthreesixpotraced]
```

We now show an example of usage (abridged). This font only has uppercase characters so one might consider duplicating these into the lowercase slots. Because we replace glyphs in the serif font used, we still have a complete font, albeit of mixed design.

```
\DEKFontP \WORD{\samplefile{knuth}}
```

This gives us a rendering that is quite readable, especially when you consider how small the bitmaps are (the red bar is the overfull box marker):

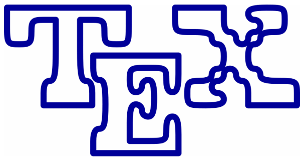
THUS. I CAME TO THE CONCLU-SION THAT THE DESIGNER OF A NEW SYSTEM MUST NOT ONLY BE THE IMPLEMENTER AND FIRST LARGE-SCALE USER: THE DESIGNER SHOULD ALSO WRITE THE FIRST USER MAN-UAL.

THE SEPARATION OF ANY OF THESE FOUR COMPONENTS WOULD HAVE HURT TEX SIGNIFICANTLY.

Just in case Don Knuth runs into this example, we need to cheat a little here and redefine the TEX logo definition:

```
\protected\def\TeX
  {\dontleavehmode
   \begingroup
   T%
   \kern-.40\fontcharwd\font`T%
   \lower.45\fontcharht\font`X\hbox{E}%
   \kern-.15\fontcharwd\font`X%
   X%
   \endgroup}
```

A real font would have proper font kerns but if needed you can set that up using the OpenType feature plug in mechanism that ConTEXt provides. A font like this can also be colored:



4 External bitmaps

A convenient way to include traced images is to use the potrace command line tool. However, we can also include grayscale single-byte PNG-encoded images:

```
\startMPcode
  path p ; p := lmt_potraced [
      filename  = "mill.png",
      criterium = 100,
  ] ;

  fill last_potraced_bounds
       withcolor "middlegray" ;
  fill p withcolor "darkgreen" ;
  draw p withcolor "darkred"
         withpen pencircle scaled 1.5 ;
  setbounds currentpicture
            to last_potraced_bounds ;

  currentpicture
    := currentpicture xsized TextWidth ;
\stopMPcode
```

This is not the best image but the photograph happens to be part of the ConTEXt distribution so why not use it. You can of course apply a different criterion and overlay a subsequent trace in a different color. The result is shown in figure 1.

In addition to the `last_potraced_bounds` path variable we also have `last_potraced_width` and `last_potraced_height` numeric available.

5 Using Lua-generated bitmaps

It is tempting to see what can be done with a bitmap generated by Lua, but let's first give a simple example. Here we register a bitmap:

```
\startluacode
local s = [[
    000000010000000000000010000000
    000000101000000000000101000000
    000001000100000000001000100000
    000010000010000000010000010000
    000100000001000000100000001000
    001000000000100001000000000100
    010000000000010010000000000010
    100000000000000110000000000001
]]

potrace.setbitmap("mybitmap",s)
\stopluacode
```

```
\startMPcode
    path p ; p := lmt_potraced [
```

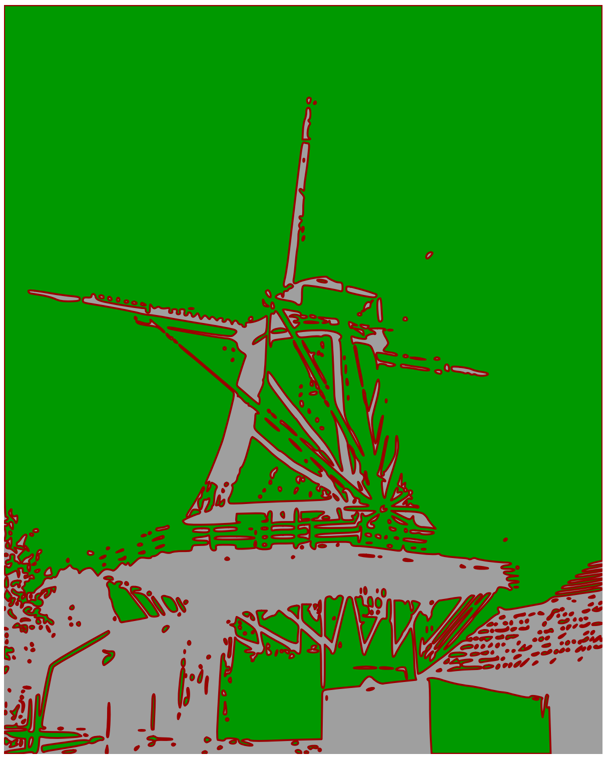
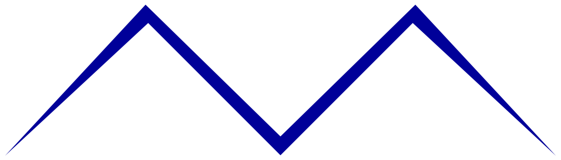


Figure 1: A traced PNG image

```
        stringname = "mybitmap",
    ] ;
    fill p ysized 2cm withcolor "darkblue" ;
\stopMPcode
```

We could play with the parameters but what we get is an outline that is supposed to be filled:

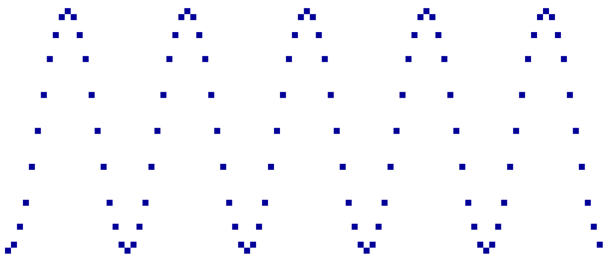


Next we create a bitmap from a dataset; in this case, we draw a function. Of course we could create some handy helpers to do this:

```
\startluacode
local d = table.setmetatableindex("table")

local t    = { }
local step = 100
local ymin = 0
local ymax = 0

local x    = 0
local dx   = 10*math.pi/step

for i=1,step do
    local y = math.round(math.cos(x)*20)
    x = x + dx
    if y > ymax then
        ymax = y
    end
    if y < ymin then
        ymin = y
    end
    t[i] = y
end

for i=1,step do
    for j=ymin, ymax do
        d[j][i] = '0'
    end
end

for i=1,step do
    d[t[i]][i] = '1'
end

for y=ymin,ymax do
    d[y] = table.concat(d[y])
end

potrace.setbitmap("mybitmap",
                  table.concat(d," ",ymin,ymax))
\stopluacode
```

```
\startMPcode
    path p ; p := lmt_potraced [
        stringname = "mybitmap",
        explode   = true,
      % tolerance = 0.5,
        threshold = 0,
      % optimize  = true,
    ] ;
    fill p withcolor "darkblue" ;
\stopMPcode
```

To what extent this is a useful application is to be decided:



Later we will see that it is less work if we stay in the first quadrant, using $(1, 1)$ as lower left corner. Here we explicitly need to provide `concat` the range because we have a negative $y_{\min}$.

It quickly gets more interesting when we use an intermediate bitmap for (a kind of) surface plots.

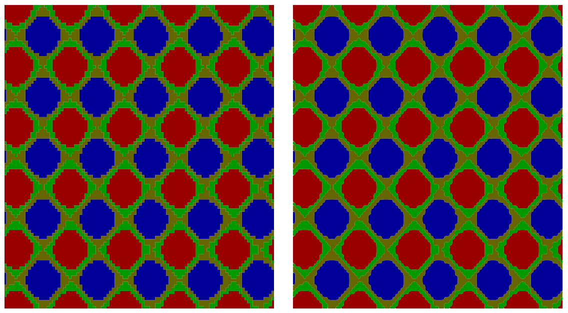
```
\startluacode
local sin, cos, pi = math.sin, math.cos, math.pi

local pp = pi/10

local function f(x,y)
    local z = sin(pp*x) + cos(pp*y)
    if z > 0.5 then
        return '1'
    elseif z > 0 then
        return '2'
    elseif z < -0.5 then
        return '3'
    else
        return'4'
    end
end
potrace.setbitmap("mybitmap",
                  potrace.contourplot(100,100,f))
\stopluacode
```

Here we use a helper that runs the given function over the maxima and collects the results in a bitmap. More such helpers will be provided when users come up with more demands.

```
\startMPcode
    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "1",
        explode    = true,
        threshold  = 0.25,
      % tolerance  = 0.1,
      % threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkred" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "2",
        explode    = true,
        threshold  = 0.25,
      % tolerance  = 0.1,
      % threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkgreen" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "3",
        explode    = true,
        threshold  = 0.25,
      % tolerance  = 0.1,
      % threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkblue" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "4",
        explode    = true,
        threshold  = 0.25,
      % tolerance  = 0.1,
      % threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkyellow" ;

    clip currentpicture to
      last_potraced_bounds enlarged -1;
    currentpicture := currentpicture
                  xysized (.45*TextWidth,4cm) ;
\stopMPcode
```

```
\startMPcode
    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "1",
        explode    = true,
      % threshold  = 0.25,
        tolerance  = 0.1,
        threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkred" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "2",
        explode    = true,
      % threshold  = 0.25,
        tolerance  = 0.1,
        threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkgreen" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "3",
        explode    = true,
      % threshold  = 0.25,
        tolerance  = 0.1,
        threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkblue" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "4",
        explode    = true,
      % threshold  = 0.25,
        tolerance  = 0.1,
        threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkyellow" ;

    clip currentpicture to
      last_potraced_bounds enlarged -1;
    currentpicture := currentpicture
                  xysized (.45*TextWidth,4cm) ;
\stopMPcode
```

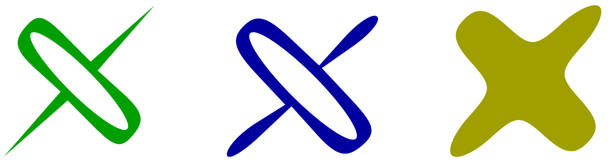
Here we are only exploring the possibilities so there is no interface like we have with contour plots. One can imagine that we provide this as a plugin, which is not that hard but whether it eventually happens depends on user demand or rainy days. So to summarize: here we generate the bitmap, we call out to potrace for a vector representation, that gets fed into MetaPost and which gives us back a result that can be converted to PDF. Of course we could go directly from potrace output to PDF if we want to, but now we get full control over the final result. In case you wonder about performance: it compiles real fast!



Some work is needed to scale the image to the proportions that reflect the input ranges but because we have a vector image that does not affect the quality of the outcome. Here we also show the effects of `tolerance` which influences the optimizing and `threshold` that determines the accuracy (number of points). In the second rendering: we use the commented values that work quite well with this kind of more mathematical images.

6 Experimenting

You can use bitmaps as a design tool but it needs a little experimenting to get the idea. Take these examples:



We started with a simple X-like symbol:

```
\startMPcode
    fill lmt_potraced [ bytes = "
        00100000001
        01010000010
        10001000100
        01000101000
        00100010000
        00010101000
        00001000100
        00010100010
        00100010001
        01000001010
        10000000100
    " ] xsized 2cm
        withcolor darkgreen ;
```

```
\stopMPcode
```

Next we started filling the shape a bit and tried to make it less spiked:

```
\startMPcode
    fill lmt_potraced [ bytes = "
        00100000001
        01010000011
        10001000100
        01000101000
        00100010000
        00010101000
        00001000100
        00010100010
        00100010001
        11000001010
        10000000100
    " ] xsized 2cm
        withcolor darkblue ;
\stopMPcode
```

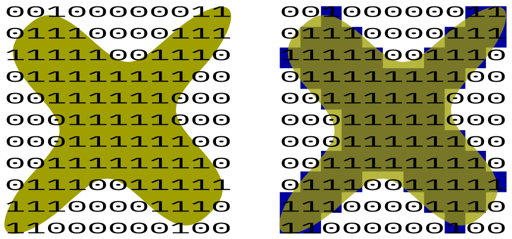
And finally we add even more ones to the bitmap. You need to fill a bitmap area in order to get an efficient fill.

```
\startMPcode
    fill lmt_potraced [ bytes = "
        00100000011
        01110000111
        11111001110
        01111111100
        00111111000
        00011111000
        00011111100
        00111111110
        01110011111
        11100001110
        11000000100
    " ] xsized 2cm
        withcolor darkyellow ;
\stopMPcode
```

If you're in doubt you can also render the bitmap, assuming that it has reasonable proportions.

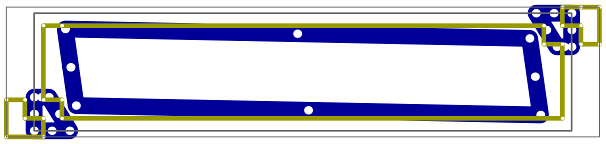
```
\startMPcode
    string s ; s := "
        00100000011
        01110000111
        11111001110
        01111111100
        00111111000
        00011111000
        00011111100
        00111111110
        01110011111
        11100001110
        11000000100
    " ;
    fill lmt_potraced [ bytes = s ]
        xsized 3cm
        withcolor darkyellow ;
    draw lmt_potraced [ bytes = s,
                        alternative = "text" ]
        shifted (.25,.25)
        xsized 3cm ;
\stopMPcode
```

The article that we mentioned in the introduction explains how potrace looks at a bits in relation to its neighbors.



You can save some runtime (and coding) by using the start-stop wrappers that keep the (intermediate) potrace object available. This permits for instance showing the 'original' polygon that serves as basis for successive steps in the library towards to the final curve(s).

```
\startMPcode
    string s ; s :=
        "0111111111111111111111111111111100
         1100000000000000000000000000000110
         1100000000000000000000000000000011
         1100000000000000000000000000000011
         1100000000000000000000000000000011
         0110000000000000000000000000000011
         0011111111111111111111111111111110";

    lmt_startpotraced [ bytes = s ] ;

        p := lmt_potraced [
            value     = "0",
            threshold = 0,
            tolerance = 0,
            optimize  = true,
        ] ;

        draw image (
            p := p shifted - center p ;
            draw p
                withpen pencircle scaled 1
                withcolor "darkblue" ;
            drawpoints p
                withpen pencircle scaled .5
                withcolor "white" ;
            draw boundingbox p
                withpen pencircle scaled .1
                withcolor "darkgray" ;
        ) ysized 3cm ;

        path p ; p := lmt_potraced [
            value   = "0",
            polygon = true,
        ] ;

        draw image (
            p := p shifted - center p ;
            draw p
                withpen pencircle scaled .25
                withcolor "middleyellow" ;
            drawpoints p
                withpen pencircle scaled .175
                withcolor "white" ;
            draw boundingbox p
                withpen pencircle scaled .05
                withcolor "darkgray" ;
        ) ysized 3cm ;

    lmt_stoppotraced ;
\stopMPcode
```



The above is just a bit of exploring the possibilities so eventually there will be a chapter on this in the LuaMetafun manual, because it is definitely fun to play with this in the perspective of MetaPost.

7 Contour plots

We end with showing how we can do rather nice contour plots and region plots with help of the potracer. Let us start with the latter type of graphics. In a recent math paper Mikael was counting nodal domains of Neumann eigenfunctions to the Laplace operator in a square. These eigenfunctions are built from cosines. One example is given by

$$\Psi(x,y) = \cos(8\pi x)\cos(3\pi y) + \cos(3\pi y)\cos(8\pi x) \ .$$

The related graphic of interest is to fill the part of the unit square where Ψ is positive. In the article, Wolfram Mathematica was used to produce this graphic, with its built-in function `RegionPlot`. The result was indeed satisfactory (Figure 2(a)).

If one looks closely at the graphics one will see that the filled regions are made up of a mesh. With potrace we instead receive one (!) path. To generate the corresponding graphic we first use Lua to define a function that takes a point as input and returns 1 if Ψ is positive at the point and 0 otherwise. We then use it to generate a 1000×1000 bitmap image of zeros and ones accordingly.

```
\startluacode
    local cos, pi = math.cos, math.pi

    local N       = 1000
    local pp      = pi/N
    local pp3     = 3 * pp
    local pp8     = 8 * pp
    local cospp8y = 0
    local cospp3y = 0

    local function f(x,y)
        if x == 1 then
            cospp8y = cos(pp8*y)
            cospp3y = cos(pp3*y)
        end
        local z = cos(pp8*x)*cospp3y
                + cos(pp3*x)*cospp8y
        if z > 0 then
            return '1'
        else
            return '0'
        end
    end

    potrace.setbitmap("mybitmap",
                      potrace.contourplot(N,N,f))
\stopluacode
```

Once this is done, we can use the result in `lmt_potraced`.

```
\startMPcode
    path p ; p := lmt_potraced [
        stringname = "mybitmap",
        value      = "1",
        threshold  = 0.25,
        optimize   = true,
    ] ;

    p := p xsized .9TextWidth ;
    fill p withcolor "darkred" ;
\stopMPcode
```

This results in Figure 2(b), which looks very similar to the one generated by Wolfram Mathematica. Note that `p` is one (disconnected) path.

We give one example of a contour plot. By a contour plot, here we mean a plot of the curve that is described as the solution to an equation $F(x,y) = 0$. With $F(x,y) = y - f(x)$ we realize that function graphs $y = f(x)$ provide a particular example, but we can also handle more complicated curves here, for example the unit circle ($F(x,y) = x^2 + y^2 - 1$).

Let us draw a part of the curve that goes under the name the Trisectrix of Maclaurin, a curve that can be used to trisect angles (named after Colin Maclaurin in 1742). We use the function

$$F(x,y) = 2x(x^2 + y^2) - (3x^2 - y^2) \ .$$

Let us construct the path and draw it.

```
\startluacode
    local N   = 1000
    local xx  = 3/N
    local yy  = 3/N
```

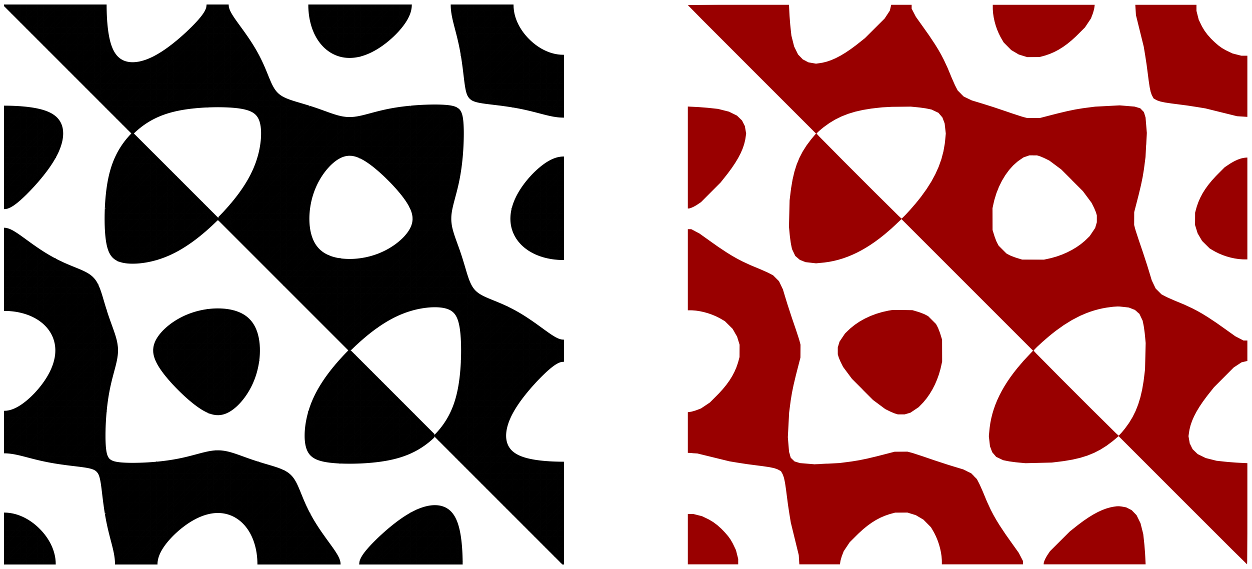


Figure 2: (a) A region plot generated by Wolfram Mathematica.
(b) The same, generated by potrace and Metafun.

```
    local function f(x,y)
        local x = xx*x - 1
        local y = yy*y - 1.5
        local z = 2*x*(x^2 + y^2) - (3*x^2 - y^2)
        if z > 0 then
            return ’1’
        else
            return ’0’
        end
    end

    potrace.setbitmap("mybitmap",
                      potrace.contourplot(N,N,f))
\stopluacode
```

```
\startMPcode
  path p ; p := lmt_potraced [
      stringname = "mybitmap",
      value      = "1",
      tolerance  = 0.1,
      threshold  = 1,
      optimize   = true,
  ] ;
  p := p xsized TextWidth ;
  draw p withcolor "darkred" ;
  drawpoints p withcolor "orange" ;
  drawpointlabels p ;
  currentpicture
    := currentpicture xsized min(8cm, TextWidth);
\stopMPcode
```

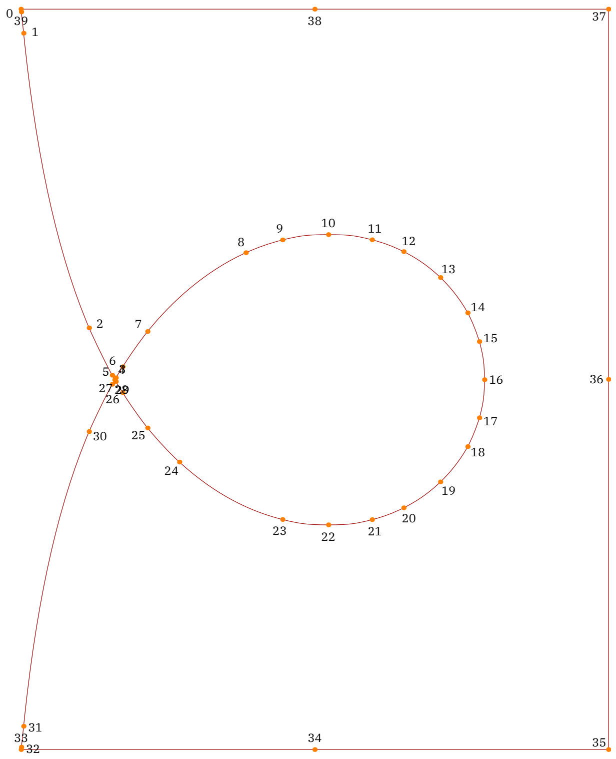


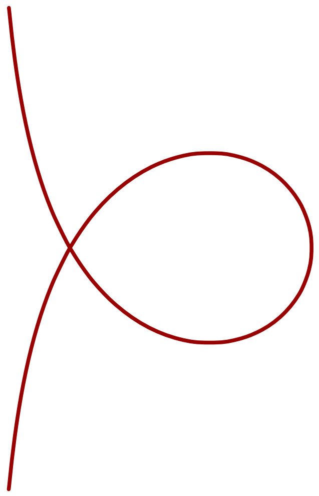
Figure 3: The Trisectrix of Maclaurin, with points and labels.

We also draw the points and the point labels. This way we can easily find out which part of the path to draw. In this case it seems that we need the first 34 points (Figure 3).

```
\startMPcode
```

```
    path p ; p := lmt_potraced [
        stringname = "mybitmap",
        value      = "1",
        tolerance  = 0.1,
        threshold  = 1,
        optimize   = true,
    ] ;
    p := subpath(0,33) of p ;
    p := p xsized 4cm ;
    draw p
        withcolor "darkred"
        withpen pencircle scaled .5mm ;
\stopMPcode
```

You need to keep the resolution (determined by the input) and therefore scaling in mind and choose the pen accordingly:

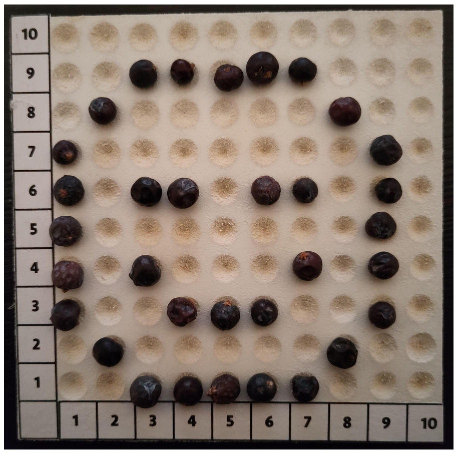


This method of drawing contour plots is efficient, and it gives, in contrast to some traditional methods, curves that look smooth, with relatively few points. It also has weaknesses, one of them being that the inequality $F(x,y) > 0$ does not always single out the curve $F(x,y) = 0$; it might be the case that the function F is positive on both sides of the curve $F(x,y) = 0$.

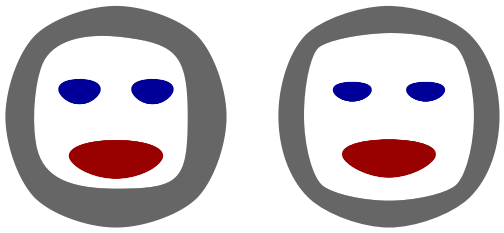
We invite the reader to be creative and play with this (to us) new toy. Have fun!

Coda

And, in the spirit of having fun, here are some final images. This first one is a photo of a “tool” that we came up with at the ConTEXt meeting. Willi Egger made a kit for the attendees, so there was the usual cutting and glueing involved. The dots are seeds.



Finally, here are dots laid out from an emoji that Mikael’s children made when we were playing with this feature. The first image has the rendering of that input, while the second “explodes” the pixels in the x direction (so columns are duplicated), resulting in a more symmetric image.



◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University
`mickep (at) gmail dot com`

Signing PDF files

Hans Hagen

Here I discuss a feature of PDF documents that TeX users can safely ignore most of the time but that is part of the package. But before we arrive at describing what it is, first a few words about PDF and the way it is used.

When PDF showed up as a format, DVI was what TeX users had to deal with. It was possible to preview the result with a DVI viewer or convert it to some format suitable for a printer. The DVI format is rather minimal and has no resources like fonts and images embedded. Basically the file only positions glyphs and rules on a canvas and the glyphs are references to external fonts. Color and similar effects have to be implemented using the `\special` primitive that puts directives for the backend driver in the file. Although technically one could embed fonts and images using specials and thereby create a self-contained file it never happened, partly because it demands a dedicated viewer and (definitely at that time) increased runtime.

At that time PostScript was one of the popular output formats. For a long time I used DVIPSONE for (robust) printer output (with outline fonts) and DVIWINDO for previewing (because it was fast, supported color, graphics and hyperlinks). The initial road to PDF was to add another step to the conversion: using Acrobat to convert a PostScript (enhanced with so-called pdfmarks) into a PDF file. Later direct DVI to PDF converters showed up alongside pdfTeX that integrated an alternative backend. We can realize that without these more direct methods TeX would not be as popular as it is now. The fact that in ConTeXt we had a rather generic (abstract) backend, where specific drivers plugged in, indicates that we didn't foresee that PDF would eventually take over.

One of the reasons PDF showed up alongside PostScript is that it removes the interpretation part from the end result. Where PostScript is a programming language and the result needs to be interpreted in the printer or in a viewer (like GhostView), PDF is a collection of related objects that express what gets rendered in what way. Often the PDF files are smaller, also due to compression (so they transfer faster), and the lack of additional processing removes a bottleneck in high speed and high resolution printing. If you look at it this way, PDF is primarily a *printer format*.

Some tools in the Adobe suite of editing programs use(d) a mix of binary and PostScript to store the state. At some point PDF became the container format, although one could find curious mixes of PostScript, PDF, even configurations stored as typeset streams, but that might have been normalized by now. So, from this perspective PDF is a *storage format*, kind of an object-oriented one.

When PDF showed up it had some rudimentary support for annotations, like hyperlinks, and also for embedding of audio and video. I'm pretty sure that TeX was among the first programs to support this. Over time more annotation types showed up, like widgets (forms), complex media and 3D, JavaScript, comments and attachments. Widgets evolved a bit and one has to play rather safe (and not use all features) because some bugs and side effects became features and there is limited support in viewers, if only because often JavaScript is assumed. Media have always been sort of a mess, especially when the straightforward annotations were dropped. We have always supported them but I consider them unreliable in the long run. Most hyperlinks are working as expected, comments (a form of PDF annotations) when used wisely also work ok, although that depends on the viewer (interfaces change). No matter what one thinks of all this, here PDF is definitely a *viewing format*. Because comments can be added, PDF files can also be used in redacting workflows.

It is also worth remembering that in the early days only Acrobat was available for viewing; there was even a version for MS-DOS. The Reader only offered basic functionality and for the real deal one needed Exchange, which didn't come free. There was a complicated scheme for shipping a special version with documents: basically that was the business model for using PDF for an on-screen reading experience. This was not a success (cumbersome as well as expensive), and when Internet access became more common, using CD-ROM for distributing documents was soon obsolete, let alone installing a related closed source and operating system dependent viewer. We'll see that with document signing, another attempt at a business model is made.

Following up on that we now arrive at two additional aspects. One is accessibility and I could spend a whole article on that. Let's stick to the observations that the idea is that rendered content can be reinterpreted for reading aloud, for reflow (sic), maybe for cut-and-paste. Personally I wonder why one would use PDF to provide adaptive accessibility, because HTML is meant for that. Distributing a structured source might be more efficient when interpretation is needed. Anyway, it's not that hard to support but we end up with a bloated PDF file, while the PDF format started out with attempts to minimize size. I understand that publishers don't

doi.org/10.47397/tb/45-1/tb139hagen-pdfsign

like to distribute sources but if this is the solution one can wonder. The second addition is to render and present text in a way that cannot be tampered with and this is where signing comes in: can we somehow mark a document such that the receiver can trust its content. More about that later, but what we can say here is that PDF has become a *distribution format*. Conforming to standards, tagging, encryption and signing all play a role in this.

No matter what usage we consider, all of them depend on reliability. Can we show and process a document in the future? Can PDF be relied upon as a long-term archival format? From that perspective, standardization has to be mentioned. Already early in the history of PDF, plugins (and additional workflows) provided the printing industry ways to check if a file was okay. One should think of color spaces being used, fonts being present, etc. Some tools manipulated the PDF, not always with the best outcome, but we leave that aside. Other tools were a bit more tolerant than might be considered healthy, for instance by ignoring a bad xref table (basically the registry of objects in a PDF file) and either fixing it or just generating one from scratch. Although Acrobat can complain or fail to open a document, on the average commercial and open source tools are tolerant enough and the lack of a proper error log means that the PDF generators don't get fixed when that still can be done. It is also good to keep in mind that there is a whole industry around PDF generation, validation, manipulation, etc. and huge money making machines are not always on the retina of, for instance, TEX users who produce PDF. Of course by now there are so many documents in PDF format around that being tolerant kind of comes with the package. Validators like VeraPDF evolve and a document that is ok today (2023) might fail the test tomorrow, and the verdict even depends on the PDF framework being used (there are options). Where TEX users can often regenerate a document from source this is not true for the majority of documents produced elsewhere.

It is also important to notice that rather soon in the history of PDF, Ghostscript became an option for viewing and at some point commercial and open source viewers showed up. Not all were perfect and even today there are differences in quality and functionality. A good test is how well cut-and-paste deals with spaces and how well a test area gets selected. The open source viewers are slow in catching up, but because the evolution of media PDF annotations isn't that stable either for most purposes viewers like SumatraPDF (Windows) or Okular (Linux) is what I use today, especially now that Acrobat has moved to the cloud. There is also some competition from browsers that show PDF. For purposes of signing (which we'll get to next) one probably has to rely on Acrobat for a while, but we'll see.

So, what does signing bring to this? Digital signatures have been around for a while. You can for instance sign a document with a certificate (similar to what secure webservers do with sites). In that case the distributed blob has security-related information as well as the content. A validating application can take the content and check if it has been tampered with. It can do so off line (with limited security) but also go online and check the embedded certificate. With signed PDF files the same is true apart from the fact that here the signature is a partial one, not embedding the data. Instead the signature is embedded in the PDF file.

Before we move on we have to stress that signing is not the same as (password protected) encryption. A signed PDF file is by default just readable, unless one explicitly encrypts the file. These processes are independent. Here we ignore encryption; suffice it to say that ConTEXt can do it, but apart from users asking for it I don't know if it ever gets applied. We discuss the process of signing in the perspective of ConTEXt, although in itself it is not bound to that macro package.

Let's first look at how text ends up in a PDF file. Take this source file, in ConTEXt-speak:

```
\startTEXpage[offset=1dk]
   some text
\stopTEXpage
```

This leads to a so-called page stream that contains this (except normally you are likely to see garbage because compression is applied, so decompress first):

```
BT
/F1 10 Tf
1.195517 0 0 1.195517 7.485099 7.616534 Tm
[<000100020003000400050006000400070006>] TJ
ET
```

We switch to a font (`Tf`) with id `F1`, set up a text transform matrix (`Tm`) and render the four plus four characters (`TJ`) indicated by their index into a (in our case subsetted) font. The `0005` is not really a character: it refers to a space. It looks unreadable but one can figure out the text by consulting the `ToUnicode` resource associated with the font as it has the mapping from the index numbers to Unicode (with comments added):

```
<0001> <0073> % s
<0002> <006F> % o
<0003> <006D> % m
<0004> <0065> % e
```

```
<0005> <0020> %
<0006> <0074> % t
<0007> <0078> % x
```

There is nothing hidden here and one can actually even change the text by changing an index in the page stream, although of course you can only use indices that are available and you also have to accept weird rendering due to the change in progression when the referenced glyph has different dimensions. More extensive tampering with the document has more severe consequences. For instance, the page object looks like this:

```
3 0 obj
<< /Length 118 >>
stream
...
endstream endobj
```

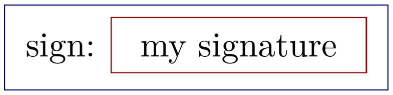
So changing the content also demands changing the `Length`. Even worse, there is an entry in the object cross reference table:

```
0000000075 00000 n
0000000244 00000 n
```

that needs to be adapted, including all following entries. So, tampering is possible but not something that is likely to happen. Nevertheless we continue as if some guard against this is needed. We now assume the following document:

```
\nopdfcompression
\setupinteraction[state=start]

\definefield[signature][signed]

\defineoverlay[signature][my signature]

\starttext
\startTEXpage[offset=1ts,frame=on,
              framecolor=darkblue]
    sign: \inframed
      [background=signature,framecolor=darkred]
      {\fieldbody[signature][width=3cm,
                             option=hidden]}
\stopTEXpage
\stoptext
```

We get a small, one page, document:


sign: my signature

This document has a widget that looks like this (some less relevant entries are omitted):

```
2 0 obj
<<
  /Type    /Annot
  /Subtype /Widget
  /FT      /Sig
  /T       <feff007300690067....074007500720065>
  /V       1 0 R
  /Rect    [ 38.445125 11.522571
             123.484489 23.471172 ]
>>
endobj
```

In principle we could add an appearance stream and decorate the widget but when adding signature support to ConTEXt I found that using a parent-kid approach, for instance, was not appreciated by some programs (I used mutool (mupdf), pdfsig (poppler), Okular and Acrobat Reader for some basic testing), so in the end the `V` key ended up in the root widget. It probably relates to fuzzy specifications, experiments with specific tool chains, non-public validation processes, etc. Round trip signing and verification seems not entirely trivial, so best to play safe.

When the value of a `Sig` widget is a string, signing is up to the viewer but when we have a dictionary the signature can be in the file. The `V` value of `1 0 R` is a reference to a dictionary with object number `1`. Here is what that value looks like when we generate this document:

```
1 0 obj
<<
  /ByteRange [ 2000000000 2000000000
               2000000000 2000000000 ]
  /Contents  <0000000000000....00000000000000>
  /Filter    /Adobe.PPKLite
  /SubFilter /adbe.pkcs7.detached
  /Type /Sig
>>
endobj
```

The `Filter` and `SubFilter` entries are sort of default, though alternatives are possible, which then requires additional information to be added and also a viewer (or validator) able to deal with it. We leave that aside. The `Contents` hex-encoded string is a placeholder for the signature and in our case is 4096 bytes long. We could compute a bogus signature and check the size instead. Here the `ByteRange` and `Contents` are actually invalid but viewers are (supposed to be) tolerant so it triggers no error. After all this widget is only consulted when signatures are checked. The general structure of the file is like this:

```
%PDF-1.7
....
1 0 obj
<< /ByteRange [ .... ] /Contents <....> .... >>
endobj
....
xref
```

The signature ends up between `<` and `>` and has to be calculated over the bytes specified by the `ByteRange` entries. Although one might think that these can be arbitrary, in practice it looks like it is best sticking to the recommendation:

```
start of file
length up to position < of contents
position after > of contents
length up to end of file
```

So basically all except the Contents value is taken into account. Because the ranges are part of that, they need to be filled in properly, something that has to be done after the PDF file is finished because only then is the size known. In ConTEXt that could be done as part of the main run, but it makes little sense because we need to adapt the file anyway. If the file is called sign-001.tex, we get this:

```
mtxrun --script pdf --sign \
       --certificate=sign-001.pem \
       --password=test sign-001
```

The script will set the byte ranges and fill in the content. It does that by making a data file and running openssl with the appropriate parameters, although with --library one can avoid the temporary file and gain a bit. Just for the record: we don't depend on that library but have only a minimal delayed binding to a few functions, with Lua wrappers so it has no impact on (compiling) the LuaMetaTEX binary. Eventually one ends up with something like this (values abridged):

```
1 0 obj
<<
  /ByteRange [ 0000000000 0000006276
               0000010375 0000000380 ]
  /Contents  <3082061a06092a....a082060b308206>
  ....
endobj
```

Verifying can be done as follows:

```
mtxrun --script pdf --verify \
       --certificate=sign-001.pem \
       --password=test sign-001
```

which reports:

```
sign pdf | signature in file 'sign-001.pdf'
           matches the content
```

while changing a byte in the trailer id results in:

```
sign pdf | signature in file 'sign-001.pdf'
           doesn't match the content
```

For verifying we can load the PDF file and use the ByteRange specification but for signing this is less trivial: when we load a PDF file we load a structure that is ignorant of the position in the file. We could use the cross reference table to find the position in the file of the object but that assumes that this table is available. So here we have two alternatives. We can write an auxiliary file (sign-001.sig) at the end of the TEX run that has the relevant information. This approach permits us to keep the PDF file simple: we reserve enough characters for the ranges and content so we can overwrite them. If the file is lacking, the sign routine tries to locate the object in the PDF from the list of widgets and once we know its number we also know where the object is in the file. This alternative adds a little overhead because at least the cross reference table has to be loaded. Whatever route we take, it is still prettier than appending additional objects to the file and basically creating a new version, which not only makes the file larger but also keeps unused objects around. Applications like mutool and Acrobat prefer that route, though, in part because they add their own appearance streams.

We now need to discuss these certificates and that is where it becomes less convenient. For testing, I use a Let's Encrypt certificate but these officially cannot be used as they are flagged as web certificates. There is (what's new here) a whole industry behind this signing. You need to get a certificate someplace and for that often have to sign up for a yearly subscription. In the worst case you get a token instead of a file and then have to set up some delegated workflow. Feeding a document into a USB token is not the most efficient of all processes, so you will find alternative solutions where you end up with a dedicated machine in a server rack. This all makes it a no-go for a low or zero budget situation. It also means that for just *printing*, *viewing* or *storing* purposes signing doesn't make that much sense: it only adds overhead.

One can argue that signing is not that robust anyway. Just like we can add a signature to a file, so can anybody. It's all about trust. When a byte in a PDF file is changed validation fails anyway so that is already a signal; we don't need to verify the certificate for that. And it's not that hard to let a user upload a file to the origin and let it validate there where the private key is known. But wait, isn't it more convenient to do that without uploading? Sure, but here are some pitfalls. First of all, who knows if a certificate is still valid? An organization has to spend quite some money on it yearly. And (even root) certificates expire so in the end the document refers to something invalid anyway, which effectively makes the document expire after some time. Saving documents and providing them again might be cheaper and also has the advantage of archiving. For long term archiving signing makes little sense anyway (expiration, cracking).

So why do we bother to add signing to ConTEXt? The answer is simple: user demand. Just like being forced to use some PDF standard, users can be forced to comply with what the organization prescribes with respect to signing, even when in the end it's just a demand, and nothing is actually done with it. So the main question is: after showing that it can be done, what eventually happens; how does the workflow look? It's comparable to tagging: it is sometimes demanded, but after that the lack of useful tools make it just a box to be ticked.

There are other alternatives to making users feel good about a document: provide a printed copy, keep the original someplace for downloading, maybe make it possible to regenerate a document from source, maybe even provide the resource. Generate a string hash and keep that available alongside the original. In the end it is all about trust, indeed.

Let's end on a positive note. Getting to know what has to end up in the file is not that trivial and as with much on the Internet, looking for solutions quickly brings you to a subset of partial and sometimes confusing answers and solutions. This is why in the end I decided to just look in the code base of `openssl` that comes with examples and eventually one can sort out something not too complex. One of the interesting observations was that the binary blob is a structured key/value sequence using technology from decades ago (1984), when data had to be transferred reliably between architectures and programming languages: Abstract Syntax Notation One (ASN.1). It makes old TEXies feel young when old tools survive beyond the modern short lifespan of fancy web technologies. I might eventually spend some more time on this, just for the fun of it.

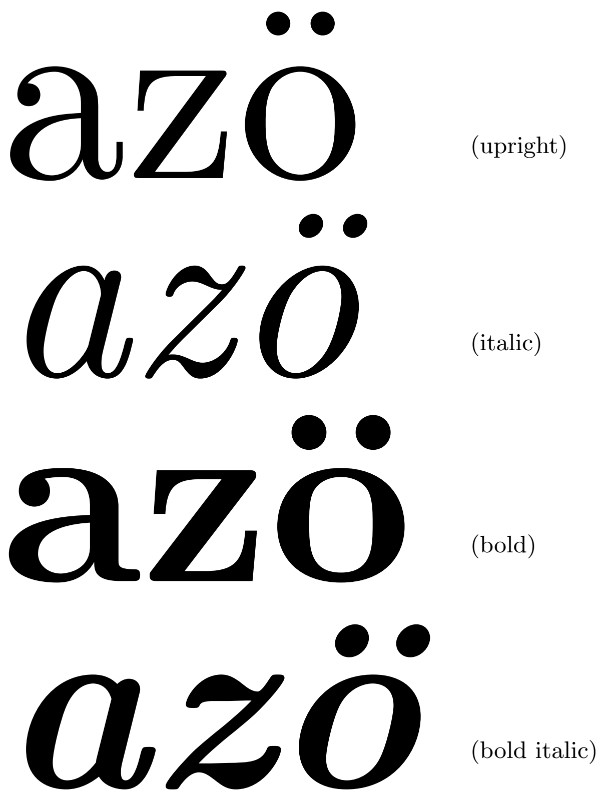
If you want to know more details: the official ISO standard on PDF has some sections on the matter; a more comprehensive summary can be found in “Digital Signatures in a PDF, Adobe Systems Incorporated, May 2012”. There is also “CDS Certificate Policy, Adobe Systems Incorporated, October 2005” but I suggest to ignore that one unless you're forced to implement the more expensive route.

Some final words on the mentioned formats. For printing and storage this feature is not needed. Nor for regular viewing, because users probably don't care that much if a manual or book is signed and it's unlikely that certificates last that long (or stay secure for that matter). But it might make sense for distributing documents with some legal meaning in the short term. In that perspective having this feature in ConTEXt makes most sense in specific workflows. But it doesn't hurt to know that TEX is still able to adapt itself to these situations.

◇ Hans Hagen
Pragma ADE

Computer Modern shape curiosities

Hans Hagen



When playing with some (upcoming) new font features in LuaMetaTEX, I overlaid regular and bold versions of Latin Modern characters. I took an ‘a’ with diaeresis as a test.

While staring at the overlays I noticed that the little hook of regular was not present in the bold variant. After displaying the whole upright alphabet, that was the only difference in shapes. In the italic shapes, the ‘z’ was a bit different. And when blown up the dots are somewhat larger in the bold. (Computer Modern is the same, naturally.)

So, the question is: how many users who can immediately recognize Computer Modern have noticed this difference in ‘a’? Another question is: did personal taste win over consistency?

We can also wonder if Latin Modern should have a few stylistic alternates, but maybe no one is willing to pay the prices in additional overhead. Of course most such details get hidden at a small 10 point size. When blown up enough, a few other interesting design details can be seen, but I leave noticing that to the reader. After all, these shapes were never meant to be seen that large.

◇ Hans Hagen
Pragma ADE

doi.org/10.47397/tb/45-1/tb139hagen-cmshapes

Radical delimiters

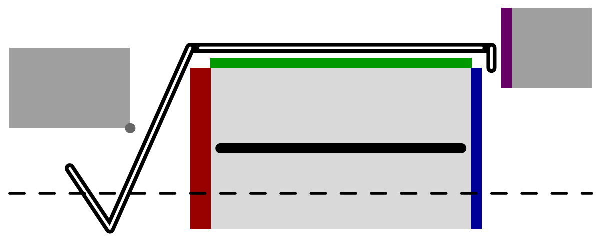
Hans Hagen, Mikael P. Sundqvist

Every TeX user who typesets math knows that left and right fences (parentheses) can grow with what they span. The same is true for the rule in a fraction, wide accents and braces (for instance) on top of a subformula. These are called horizontal and vertical extensibles. There are also special extensibles like integrals, sums and products. Integrals sometimes can grow indefinitely but the latter two come in a limited set of variants. Even (for instance) parentheses start with stepwise larger variants before we end up with an extensible.

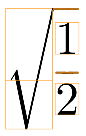
A math radical is also an extensible: the left part of this symbol can grow but in traditional TeX the bar at the top is a rule. There is no real concept of a two-dimensional extensible and for reasons unknown to us OpenType didn't bother to add them. That would also introduce a right part being supported. In the next abstraction we show a bunch of properties that we have to deal with.

Because we have no hope that this will become available we've rolled our own. The middle piece can be a glyph like with any extensible and we support a right piece. For this the engine was adapted. But it's not enough. When a variant grows, the angle might change slowly till we go upwards. In many fonts the number of variants is not enough to accommodate proper rendering; think of plain symbols, symbols with a script, fractions, fractions that themselves contain symbols with scripts. It can be hard to come up with a configuration that works well for each of them when we lack variants.

The next graphic shows what we're dealing with. Here the radical shape is made from a single left, repeated middle and a single right piece but the left and right ones can also be made from pieces, when they are upright.



When a variant is sufficiently sloped, there is a danger that it will clash with the content, so we need some kerns that depend on the shape. In the picture above, they're shown as the vertical bars at left and right under the radical in red and blue, respectively (grayscaled for *TUGboat*), and hopefully also visible in this example:



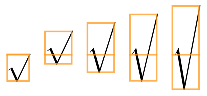
This is a character-specific property. We already have the distance between content and top as a parameter (horizontal bar at top, in green) and of course this is different for text and display math. Then we have the degree of the radical, which has its own vertical positional parameter but again we need something per size. Because we have only a shared parameter the leftmost part of the symbol is always the same, although we could abuse some depth trickery here. So we need proper anchors so that the degree can stick out to the left of that anchor (gray dot) instead of using some heuristic (if at all: we can also overlap). This shape-dependent margin is not to be confused with margins that we add in ConTeXt, at the left and/or right, as well as enforced by struts.

We can even think of kerning between the radical left symbol and the first one seen in the content. This is kind of complicated because we get a chicken-egg situation as the symbol depends on the content and if that becomes wider we need to recalculate the assembly. So for now there is no extra kern (after the red) even if we do support kerning in some cases.

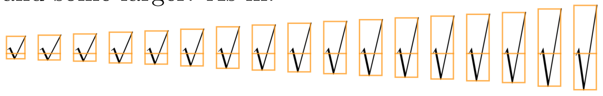
The radical as whole also has properties, for instance the right symbol can demand some top kern (magenta). Actually a prescript will kern before the left symbols but is not needed in case of a degree and given the white areas there, such a kern is unlikely. And the user might also expect the radicals in a formula made from more than one radical to have the same size. There is also the math axis (heavy black line) to deal with because the symbol gets vertically centered over the content.

So we have quite a few extra shape-dependent properties to deal with: margins, offsets and (corner) kerns. We also need multiple passes in order to meet demands like comparable sizes and calculating content dimensions that are needed for the sizing. Keep in mind that traditional TeX is eight bit and assumes a single extensible font (number three of the hard coded four family setup) and the 256 slots have to be distributed across variants and sizes and so TeX can provide only a limited solution space here.

All this (plus some more) is supported in LuaMetaTeX but it only works out well if the macro package provides the information that is lacking in the fonts, which is yet another reason why we have companion fonts (adding sizes and fixing inconsistencies which are hard to tweak) as well as math font goodie files that add the information needed. It goes without saying that the authors spent a considerable amount of time on getting all this right. For example, we found out that the four variants of the radical in Latin Modern:



could benefit from extra sizes, some intermediate, and some larger. As in:



Compare the output of a few typical radicals:

$$\sqrt{2} + \sqrt{2^2} + \sqrt{\frac{1}{2}} \qquad \sqrt{2} + \sqrt{2^2} + \sqrt{\frac{1}{2}}$$

original companion

It is fair to mention that in ConTEXt we do use struts inside radicals and fractions to enforce consistency, and therefore the outcome of these examples might look different in other macro packages.

In principle all math symbols have these extra properties attached but their usage differs, so for instance in accents the margins are around the (wide) accent. With radicals the top and bottom margins are ignored but as we progress they might get some meaning. In some cases the implementation is less straightforward, for instance in a 'binop' we have a fraction with built-in fences so there we need to carry over the kerns that come with the chosen fences.

This all means that the math engine is more complex so it starts making sense to consider removing the traditional code paths: no new old school math fonts are likely to show up, and the ones that we had have acquired OpenType implementations by now anyway.

Sometimes, when we see what users try to compensate for, or ask for fixes about (on Stack Exchange, for instance), we wonder if this is a recent observation. After all, nothing like the above made it into OpenType math and fonts. Maybe observations get lost after some quick fix instead of being accumulated in some proposal, or maybe nothing got fixed anyway. We never get (or see) reports from editors (of journals) and none of them seem to follow developments like this (or we'd have noticed).[1] Of course much goes unnoticed when seen in print (at desktop or office printer resolution), but can be seen when proofing or reading on screen.

◇ Hans Hagen
◇ Mikael P. Sundqvist

[1] An exception is *TUGboat*'s Karl Berry who gives us valuable and inspiring feedback.

The Treasure Chest

These are the new packages posted to CTAN (`ctan.org`) from October 2023–April 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*.

A few entries which the editors subjectively believe to be especially notable are starred (`*`); of course, this is not intended to slight the other contributions.

◇ Karl Berry
`https://tug.org/TUGboat/Chest`
`https://ctan.org/topic`

biblio

`chicagolinks` in `biblio`
Annotated bibliographies including DOI links.

`iran-bibtex` in `biblio/bibtex/contrib`
Implementation of the *Iran Manual of Style Citation Guide* for BIBTEX.

fonts

`junicodevf` in `fonts`
Variable font family for mediaevalists. (See article on pp. 12–17.)

`lato-math` in `fonts`
Lato-based OpenType math font.

`ysabeau` in `fonts`
Garamond-like design with a low-contrast sans serif.

graphics

`pmdraw` in `graphics`
Draw elements of the partition monoids.

graphics/pgf/contrib

`argumentation` in `graphics/pgf/contrib`
Create abstract argumentation frameworks.

`pictochrono` in `graphics/pgf/contrib`
Inline chronometer pictograms, with durations.

`polyhedra` in `graphics/pgf/contrib`
A Ti*k*Z package for drawing polyhedra.

`thematicpuzzle` in `graphics/pgf/contrib`
Add horizontal banners in a puzzle style.

`tikzdotncross` in `graphics/pgf/contrib`
Defining/marking coordinates and crossing paths with jumps.

`graphics/pgf/contrib/tikzdotncross`

`tikzquads` in `graphics/pgf/contrib`
Shapes designed to be used with CircuiTi*k*Z.

`trivialpursuit` in `graphics/pgf/contrib`
Generic Trivial Pursuit board game.

`twoxtwogame` in `graphics/pgf/contrib`
Visualize 2×2 normal-form games.

info

`typstfun` in `info`
Typst function equivalents of LaTeX commands.

macros/generic

`calcfrac` in `macros/generic`
Calculate value of an expression containing fractions.

macros/latex/contrib

`affilauthor` in `macros/latex/contrib`
Tag author and affiliation information in a key–value style.

`amnestyreport` in `macros/latex/contrib`
Class for Amnesty International.

`beautynote` in `macros/latex/contrib`
Book design with several chapter and page styles.

`chemformula-ru` in `macros/latex/contrib`
Using `chemformula` with `babel-russian`.

`cidarticle` in `macros/latex/contrib`
Class for *Commentarii informaticae didacticae*.

`cleveref-forward` in `macros/latex/contrib`
Forward-referencing functionality for `cleveref`.

`cjs-rcs-article` in `macros/latex/contrib`
Class for *The Canadian Journal of Statistics*.

`coloredbelts` in `macros/latex/contrib`
Insert colored belts as vector images.

`coloredtheorem` in `macros/latex/contrib`
A colorful boxed theorem environment.

`contract` in `macros/latex/contrib`
Typeset formalized legal documents such as contracts, statutes, etc.

`cs-techrep` in `macros/latex/contrib`
Technical report style, similar to IEEE.

`dashrulex` in `macros/latex/contrib`
Draw dashed rules.

`decimalcomma` in `macros/latex/contrib`
Comma for decimal numbers.

`didactic` in `macros/latex/contrib`
Tools for writing teaching material: semantic environments, slides and notes from the same source, code and output side by side, etc.

`didec` in `macros/latex/contrib`
Fixed-point arithmetic with two decimal places, for financial transactions.

`fadingimage` in `macros/latex/contrib`
Full width fading pictures at the top or bottom of a page.

`fontscale` in `macros/latex/contrib`
Flexible interface for setting font sizes.

`freealign` in `macros/latex/contrib`
Align math formulas in different lines.

`genealogy-profiles` in `macros/latex/contrib`
Genealogical profiles for LaTeX.

`heria` in `macros/latex/contrib`
Class for Horizon Europe RIA and IA grant proposals. (See article on pp. 59–64.)

`iaria` in `macros/latex/contrib`
Class for IARIA scholarly publications, including citation style.

`iaria-lite` in `macros/latex/contrib`
IARIA support except for citation style.

`ipsum` in `macros/latex/contrib`
Insert multilingual placeholder text.

`jsonparse` in `macros/latex/contrib`
Parse JSON data from files or strings into token variables.

`latex2pydata` in `macros/latex/contrib`
Write data to file in Python literal format.

`litebook` in `macros/latex/contrib`
Fresh cover and chapter design for books.

`litesolution` in `macros/latex/contrib`
Light design for solutions of test papers.

`litetable` in `macros/latex/contrib`
Class schedules with colorful course blocks.

`medmath` in `macros/latex/contrib`
Improve the `mediummath` option in `nccmath`.

`nameauth` in `macros/latex/contrib`
Name authority mechanism for consistency in body text and index.

`notebeamer` in `macros/latex/contrib`
Template for presentations on notepaper.

`odesandpdes` in `macros/latex/contrib`
Optimizing workflow involving odes and pdes.

`pdfannotations` in `macros/latex/contrib`
Annotate PDF slides.

`pgfkeysearch` in `macros/latex/contrib`
Find keys in a given `pgfkeys` path recursively, unlike `pgfkeysvalueof`.

`pynotebook` in `macros/latex/contrib`
Present code, along with execution, as in a Jupyter notebook.

`q-and-a` in `macros/latex/contrib`
Typesetting Q&A-style conversations.

`randexam` in `macros/latex/contrib`
Make an exam paper and randomized variants.

`regulatory` in `macros/latex/contrib`
Flexible drafting of legal documents, especially in Dutch.

`reptheorem` in `macros/latex/contrib`
Replication of theorem environments, including across documents.

`responsive` in `macros/latex/contrib`
Responsive design methods for LaTeX.

`sfee` in `macros/latex/contrib`
LaTeX class for the *Smart Factory and Energy Efficiency* journal.

`sim-os-menus` in `macros/latex/contrib`
Insert a 'terminal' or 'context menu' or 'viewer' image, as from an OS.

`sjtutex` in `macros/latex/contrib`
LaTeX classes for Shanghai Jiao Tong University.

`tblr-extras` in `macros/latex/contrib`
Libraries for `tabularray` for `caption` and `babel` compatibility.

`thmlist` in `macros/latex/contrib`
Adding new theorem-like environments.

`tikzquests` in `macros/latex/contrib`
A parametric questions' repository framework.

`tutodoc` in `macros/latex/contrib`
Typeset tutorial-like documentation.

`udepcolor` in `macros/latex/contrib`
University of Piura colors.

`undar-digitacion` in `macros/latex/contrib`
Musical fingering diagrams for flute, recorder, sax, et al.

`useclass` in `macros/latex/contrib`
Load classes as packages; developed for `l3doc`.

`vectorlogos` in `macros/latex/contrib`
Some logos in vector format, mostly TeX-related.

`verifycommand` in `macros/latex/contrib`
Verify definitions are unchanged, such as before patching.

`weiqi` in `macros/latex/contrib`
Use LaTeX3 to typeset Weiqi (Go).

`xkeymask` in `macros/latex/contrib`
Extension of `xkeyval` to dynamically (un)mask options.

m/l/c/beamer-contrib/themes

`beamerthemeconcrete` in `m/l/c/b-c/themes`
Collection of flat beamer themes.

`moloch` in `m/l/c/b-c/themes`
Updated version of the Metropolis theme.

macros/luatex/latex

`autotype` in `macros/luatex/latex`
Automatic language-specific typography: weighted hyphenation, et al., for German.

`gitinfo-lua` in `macros/luatex/latex`
Display `git` project information.

`ideavault` in `macros/luatex/latex`
Idea (concept) management, e.g., for handbooks.

`longmath` in `macros/luatex/latex`
Nested delimiter groups extending over multiple array cells or lines.

`lua-placeholders` in `macros/luatex/latex`
Specifying external values for insertion into templates. (See article on pp. 65–76.)

macros/unicodetex/latex

`emotion` in `macros/unicodetex/latex`
Make emojis easier to typeset.

macros/xetex/latex

`quran-en` in `macros/xetex/latex`
English translation extension to the `quran` package.

`quran-id` in `macros/xetex/latex`
Indonesian translation extension to `quran`.

support

* `l3sys-query` in `support`
System queries for LaTeX using Lua.

`latex-dependency-grapher` in `support`
Java program to visualize the dependencies of LaTeX files, using GraphViz.

`ppmcheckpdf` in `support`
Convert PDF to PNG and compare PNG files after `l3build`.

`texblend` in `support`
Compile segments of LaTeX documents.

Production notes

Karl Berry

We publish a “complete” PDF with each issue of *TUGboat*, as the file `tb`⟨*nnn*⟩`complete.pdf`. These `complete.pdf` files start with the back cover table of contents, then the inside front cover, then all the interior pages of the issue, and end with the inside back cover, the “contents by difficulty”. The printed front cover is omitted, since it may include large images that would greatly increase the file size.

For several years (since vol. 38, no. 3, in 2017), the page numbers on both tables of contents have been internal links to the given article within the issue, for easy navigation to a particular article. (Thanks to Frank Mittelbach for prodding us to implement that.) With that feature, however, the external hyperlinks (to web pages, etc.) within the document have been lost within the `complete.pdf`s.

This is because the `complete.pdf` is necessarily created by concatenating various PDF files. (We cannot create it in a single TeX run because different articles require different engines, among other reasons.) Essentially any PDF tool will do the concatenation, but they all lose the links within the included file. I tried pdfTeX itself, Ghostscript, `qpdf`, `mupdf`, and plenty more.

This is not surprising, since links as such are not a basic concept of the PDF format: external links are so-called annotations that define an action for a rectangular area on a page, and internal links go to objects within the PDF file. Normally, it does not make sense to preserve either of these when including one PDF in another.

The `newpax` package by Ulrike Fischer (`ctan.org/pkg/newpax`), updating Heiko Oberdiek’s `pax`, can preserve links by use of external code (written in Lua for `newpax` and Java for `pax`). The problem for me was that the associated `newpax.sty` requires LaTeX, and all of *TUGboat*’s table of contents processing is written in plain TeX. (The original *TUGboat* code was written before LaTeX existed, and we are still using it, largely unchanged.) I was not enthused about rewriting the entire process in LaTeX.

So I asked on the development mailing list, `ntg-pdftex`. Taco Hoekwater (thanks Taco) pointed out that ConTeXt supported keeping “interaction” elements, such as links, when including a PDF. This was a step forward, but unfortunately the internal links from the table of contents were still lost.

Ultimately, Hans Hagen came to the rescue, implementing the exact feature needed, in his LMTX engine and ConTeXt. Thanks so much, Hans! The invocation looks like:

```
context --extra=copy --template tbcomplete.lua
mv context-extra.pdf tbNNNcomplete.pdf
```

where the Lua “template” file looks approximately like this (for *TUGboat* 44:2):

```
return { list = {
  {
    filename    = "toclinks.pdf",
    first       = 1,  --cover1
    last        = 2,  --cover2
    interaction = "all", pageoffset  = 0,
  },
  {
    filename    = "issue.pdf",
    first       = 1,    --interior of issue
    last        = 176,
    interaction = "all", pageoffset  = 0,
  },
  {
    filename    = "toclinks.pdf",
    first       = 179,  --cover3, first page
    last        = 180,  --cover3, second page
    interaction = "all", pageoffset  = 0,
  },
}}
```

Here, `toclinks.pdf` is the PDF made with the interior toc links and `issue.pdf` is the full issue (176 pages in this case) with active external links. This issue was so long that the contents by difficulty printed on the inside back cover (“cover3”) spilled over to an additional page. (The `pageoffset` parameter allows for skipping pages, which we don’t need here. There is plenty of other functionality, too.)

With this new functionality available, I have remade the `complete.pdf` files back to vol. 40, no. 2, so they now contain both tables of contents internal links and external hyperlinks within the issue. (Before that, the issue pages did not contain (m)any hyperlinks, so there’s little benefit.)

I also took the opportunity to extend the internal links to also be active on the author names and titles, as well as the starting page numbers, as of issue vol. 44, no. 2.

Examples of the source code are available in the *TUGboat* source repository at `tug.org/svn/tugboat/trunk/covers` (or its mirror at `github.com/TeXUsersGroup/tugboat`). The main files are `tbcomplete.lua` for the Lua template file above, and `tbcomplete.tex` for the plain TeX that adds the links to the tocs. Although the code is not directly runnable in other environments, it might serve as a basis for those interested.

Thanks again Hans!

◇ Karl Berry
`github.com/TeXUsersGroup`

Book review: *Shift Happens* by Marcin Wichary

Barbara Beeton, Karl Berry, Boris Veytsman

Marcin Wichary, *Shift Happens*, October 2023, 1216+160 pp., 2+1 vols., ISBN 9798985873900. `https://shifthappens.site`

The book *Shift Happens* by Marcin Wichary, mentioned in Barbara's column in the last *TUGboat*,[1] was published near the end of 2023. The book tells the story of keyboards from the earliest typewriters (ca. 1870) to the virtual keyboards made of pixels in our pockets today, in approximately 1200 pages and 1300 photographs (many taken by the author). Notwithstanding the profusion of photos, it is not a "coffee table" book; the text tells the story.

There is a wealth of information about the book and keyboards in general on its web site: a commentary on the text, essays on related topics, the run of the newsletter that the author published during the book's production, and several keyboard games (e.g., "make your own dvorak hands"). The book is strikingly designed and typeset by the author. He also oversaw the final production process, visiting the printing plant (in Maine) in person, and has plenty of interesting things to say about the whole process. The Kickstarter campaign updates tell much of the story.[2]

Before publication, the Museum of Printing (MoP) hosted a panel with the author and other typographic luminaries. The discussion, more than an hour long, was recorded, and is posted on MoP's YouTube channel in three parts.[3]

For those interested in the nuts and bolts of publication, Glenn Fleishman, the book's editor and manager of print production, crowdfunding, and fulfillment, wrote a terrific essay about the funding campaign and how Kickstarter works on the back end: "How We Crowdfunded $750,000 for a Giant Book about Keyboard History".[4]

Along with the book, Wichary worked with type designer Inga Plönnings to create Gorton Perfected No. 2, an OpenType font based on the design originating with the George Gorton Machine Company of Racine, Wisconsin, ca. 1900. The original has been pervasively used on engraved signs of all kinds, and later adapted to keycaps. A sample is shown below, and Marcin created a 96-page specimen booklet with many in situ examples.[5]

The book is completely sold out, but you can express interest in participating in another print run, in the event that one happens, on the web site.

Congratulations, Marcin!

◇ Barbara Beeton, Karl Berry, Boris Veytsman
`tug.org/books`

[1] `tug.org/TUGboat/tb44-3/tb138beet.pdf`

[2] `kickstarter.com/projects/mwichary/shift-happens`

[3] `youtube.com/watch?v=HrlcFB_Q_UE`
`youtube.com/watch?v=-8a4XwOtd9k`
`youtube.com/watch?v=SWfDrcdXfaM`

[4] `glennf.medium.com/how-we-crowdfunded-750-000-for-a-giant-book-about-keyboard-history-c30e24c4022e`

[5] `shifthappens.site/gorton-perfected-specimen.pdf`

On November 14, 1885, Senator & Mrs. Leland Stanford called together at their San Francisco mansion the 24 prominent men who had been chosen as the first trustees of The Leland Stanford Junior University. They handed to the board the Founding Grant of the University, which they had executed three days before. This document—with various amendments, legislative acts, and court decrees—remains as the University's charter. In bold, sweeping language it stipulates that the objectives of the University are "to qualify students for personal success and direct usefulness in life; and to promote the publick welfare by exercising an influence in behalf of humanity and civilization, teaching the blessings of liberty regulated by law, and inculcating love and reverence for the great principles of government as derived from the inalienable rights of man to life, liberty, and the pursuit of happiness." ¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés? (¡THE DAZED BROWN FOX QUICKLY GAVE 12345–67890 JUMPS!) №1@2. €100.00=$108.74=10874¢=₤85.46. ←↑→↓↵⇤⇥⇧⌃⌘⌥⌦⌫⇪

doi.org/10.47397/tb/45-1/tb139reviews-wichary

Die TEXnische Komödie 3/2023–1/2024

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`dante.de`).

Die TEXnische Komödie 3/2023

Luzia Dietsche, Eindrücke von Bonn, Mitgliederversammlung und TUG 23 [Impressions from the DANTE e.V. members' meeting and TUG 23]; pp. 7–11

This year, the general meeting of DANTE e.V. took place in Bonn, just before the TUG conference.

Keno Wehr, LATEX und Schulphysik 3: Messwertdiagramme [LATEX and Physics in School 3: Diagrams for measured values]; pp. 12–25

The third installment of this series covers various diagrams useful in physics education.

Christian Böttger, LATEX per `pandoc` mit Markdown füttern [Feeding LATEX with Markdown via `pandoc`]; pp. 26–36

For many, Markdown has become an alternative to common typesetting engines or word processors. In this article the author shows how LATEX can be fed by Markdown and the Pandoc converter.

Pfarrer HG Unckell, Erfahrungsbericht zum Einsatz von TEX im Alltag eines Gemeindepfarrers [Experiences of the usage of TEX in the daily work of a parish priest]; pp. 36–37

A short article on how LATEX can be used by a German parish priest.

Henning Hraban Ramm, ConTEXt kurz notiert [Short notes on ConTEXt]; pp. 38–41

Frank Mittelbach, LATEX-News – Issue 37, Juni 2023 [LATEX News — Issue 37, June 2023]; pp. 42–54

[Published in *TUGboat* 44:2; translated by Thomas Demmig. Posted at `latex-project.org/news/latex2e-news`.]

Jürgen Fenn, Neue Pakete auf CTAN [New packages on CTAN]; pp. 55–58

Uwe Ziegenhagen, *LATEX Cookbook, 1. Auflage* — Buchrezension [*LATEX Cookbook, 1st edition* — book review]; pp. 59–60

A review of the first edition of Stefan Kottwitz's LATEX cookbook. (English translation published in this issue.)

Uwe Ziegenhagen, *LATEX Graphics with TikZ* – Buchrezension [*LATEX Graphics with TikZ* — book review]; pp. 61–62

A review of the first edition of Stefan Kottwitz's book on Ti*k*Z.

Die TEXnische Komödie 4/2023

Volker RW Schaa, Protokoll der 65. Mitgliederversammlung von DANTE e. V. am 13. Juli 2023 im Collegium Leoninum, Bonn [Minutes of the 65th user group meeting of DANTE e. V. on July 13th, 2023 at Collegium Leoninum in Bonn]; pp. 7–14

The official minutes of the user group meeting in Bonn.

Doris Behrendt, Bericht der Schatzmeisterin für das Jahr 2022 [Report of the Treasurer for the year 2022]; pp. 15–22

Nadia Kwast and Marei Peischl, Bericht der Kassenprüfer für das Jahr 2022 [Report of the internal auditors for the year 2022]; pp. 22–26

Martin Sievers, DANTE e.V. sucht Veranstalter für Tagungen [DANTE e.V. seeks meeting organizers]; p. 27

Keno Wehr, LATEX und Schulphysik 4: Messinstrumente [LATEX and Physics in School 4: Measuring devices]; pp. 28–37

The fourth article of this series focuses on the display of measuring devices in LATEX documents.

Ralf Mispelhorn, Darstellen von Mengen-Operationen mit Ti*k*Z [Showing set operations in Ti*k*Z]; pp. 38–44

To visualize the set operations of the Python scripting language, diagrams were created with Ti*k*Z with the help of clipping functionality and the *even-odd* fill rule. Originally a lecture at the Duale Hochschule Horb.

Günter Rau, Serienbriefe im Adressfeld frankieren [Adding postage to the address label of bulk mail]; pp. 44–51

With the help of the letter package `scrlttr2` and the CSV package `csvsimple`, it is easy to create a template with which you can send individual and form letters which are automatically franked.

Werner Lemberg, Verbesserung des `@code` Befehls in Texinfo mit LuaTEX [Improvements to the `@code` command in Texinfo with the help of LuaTEX]; pp. 52–64

This article describes how someone using the `@code` command in Texinfo with LuaTEX can intelligently improve line breaking at the symbols `-` and `_`.

Henning Hraban Ramm, ConTEXt Meeting 2023 [ConTEXt Meeting 2023]; pp. 65–71

Notes from the ConTEXt Meeting 2023.

Jürgen Fenn, Neue Pakete auf CTAN [New packages on CTAN]; pp. 71–77

Die TEXnische Komödie 1/2024

Martin Sievers, Grußwort [Greeting]; pp. 4–5

Introductory words from the DANTE president.

Martin Sievers and Thomas Hilarius Meyer, Einladung zur Frühjahrstagung 2024 und 66. Mitgliederversammlung von DANTE e.V. im Goethe-Nationalmuseum in Weimar [Invitation to the spring conference 2024 and 66th general meeting of DANTE e.V. in the Goethe National Museum in Weimar]; pp. 6–7

The spring conference of DANTE will take place from April 4th to April 6th at the Goethe National Museum in Weimar.

Martin Sievers and Thomas Hilarius Meyer, Beiträge gesucht [Call for presentations]; p. 8

Call for presentations for the spring conference.

Martin Sievers, DANTE Tasse [DANTE cup]; p. 9

The DANTE cup is available.

Keno Wehr, LATEX und Schulphysik 5: Mechanik und Astronomie [LATEX and Physics In School 5: Mechanics and astronomy]; pp. 10–23

The fifth part of the series of articles on school physics deals with graphics representations from the fields of mechanics and astronomy. It presents the MetaPost `fiziko` package as well as the `pst-pulley` and `pst-solarsystem` packages.

Keno Wehr, Sprachspezifische Typographie mit `autotype` [Language-specific typography with `autotype`]; pp. 23–44

The LuaLATEX `autotype` package can be used to meet language-specific typographical requirements automatically. One can choose one syllable hyphenation method with differently weighted separation points. At the moment only the German language (old and new spellings) is supported.

Henning Hraban Ramm, ConTEXt kurz notiert [ConTEXt news]; pp. 44–46

There's a lot going on in the ConTEXt world that's worth mentioning, but does not justify an entire article.

Frank Mittelbach, LATEX-News – Issue 38, November 2023 [LATEX News — Issue 38, November 2023]; pp. 47–55

[Posted at `latex-project.org/news/latex2e-news`.]

Jürgen Fenn, Neue Pakete auf CTAN [New packages on CTAN]; pp. 55–60

Jerzy Ludwichowski, The 29th GUST TEX conference — Composed thoughts; pp. 61–62

This year's theme, "Composed thoughts", can be interpreted in several ways. First, it can be seen as a reference to the process of creating a document with TEX, during which thoughts can be separated from the final graphical form into which they will be composed. Second, the theme can be seen as a reference to TEX's potential for "composing" thoughts: ways of conveying complex ideas, structuring arguments, and illustrating them with specialized notations and diagrams. Finally, this year's theme can be seen as an invitation to reflect on where the order, structure, and "being well composed" ends, and in what situations the tool we use ceases to be elegant, and chaos creeps into files.

We invite you to participate in this year's conference and encourage you to show your "composed thoughts", as well as those more chaotic because, at BachoTEX, ideas are born between heads.

[Received from Uwe Ziegenhagen.]

La Lettre GUTenberg 51, 2023

La Lettre GUTenberg is a publication of GUTenberg, the French-language TeX user group (`gutenberg-asso.org`)

La Lettre GUTenberg #51

Published November 12, 2023.
`doi.org/10.60028/lettre.vi51`

Patrick Bideault, Éditorial [Editorial]; pp. 1–2

François Druel, Procès verbaux [Reports of board's meetings]; pp. 2–7

Journée GUTenberg 2023, le 18 novembre, en présentiel [GUTenberg Day and General Assembly 2023]; pp. 7–8

The day's program includes two lectures, by Alain Matthes and Bastien Dumont and a presentation of the new French FAQ, by Denis Bitouzé.

Maxime Chupin, Bilan moral : janvier — novembre 2023 [Moral assessment: January — November 2023]; pp. 9–13

François Druel, Rapport financier 2023, budget 2024 et proposition de cotisation [Financial report for the year 2023, budget for 2024 and fees proposal]; pp. 14–18

Maxime Chupin, Exposés mensuels sur (LA)TeX et autres logiciels, l'aventure est lancée [Monthly conferences: a successful launch]; pp. 19–21

GUTenberg has offered online conferences every month since last summer.

Patrick Bideault, Denis Bitouzé, Maxime Chupin & Yvon Henel, Et maintenant, une bonne *vieille* veille technologique ! [Technology watch]; pp. 21–33

70 new CTAN packages, June–October 2023.

Victor Sannier, TeX et LaTeX hors ligne : TUG 2023 à Bonn [TeX and LaTeX offline: the TUG 2023 conference in Bonn]; pp. 33–35

Report about the conference by Victor Sannier, whose conference about his typeface work was funded by GUTenberg.

Barbara Beeton, Ce que tout débutant (LA)TeX devrait savoir [What every (LA)TeX newbie should know]; pp. 36–48

A translation of the article published in *TUGboat* volume 44:2, 2023.

Cédric Pierquet, Brève introduction à une compilation assistée, grâce à arara [A short introduction to assisted compilation with arara]; pp. 48–58

Maxime Chupin, La fonte du numéro : Arsenal [This issue's font: Arsenal]; pp. 59–62

Arsenal is a font created by Andrij Shevchenko. It won the Ukrainian Type Design Competition 'Mystetsky Arsenal' in 2011 and was recently packaged for LaTeX by Boris Veytsman.

Patrick Bideault & Maxime Chupin, En bref [At a glance]; pp. 63–65

Short news items about the analysis of the typographic composition of a novel, a few wallpapers, a video and more.

[Received from Patrick Bideault.]

Zpravodaj 2024/3–4

Zpravodaj is the journal of CSTUG, the TEX user group oriented mainly but not entirely to the Czech and Slovak languages. The full issue can be downloaded at `cstug.cz/bulletin`.

Vít Starý Novotný, Úvodník [Editorial]; pp. 61–62

The editorial presents an overview of the articles from this issue and announces TUG 2024, which will be held in Prague.

Vít Starý Novotný, CSTUG na konferenci TUG 2023 [CSTUG at the TUG 2023 conference]; pp. 63–65

A report on the participation of CSTUG members at TUG 2023 in Bonn.

Jan Šustek, Generování dokumentovaného zdrojového souboru po blocích v TEXu [On generating documented source code by blocks in TEX]; pp. 66–101

This paper concerns writing programs and their documentation. We show author's package gensrc running on OPmac, which allows writing both program code and its documentation in one TEX file. We also show more possibilities and applications of this package.

Jan Šustek, Jak umožnit stránkový zlom uvnitř vložených obrázků [How to enable page breaks in embedded images]; pp. 102–110

This paper defines and describes TEX macros for inserting objects which are so tall that the page breaking is difficult. These objects can be images, text examples or generally a content of a box. The macros insert the object at the current position and they allow a page break in the middle of the object.

Vít Starý Novotný, Markdown 3: Co je nového a co se chystá? [Markdown 3: What's new, what's next?]; pp. 111–124

The Markdown package for TEX has provided an extensible and format-agnostic markup language for the past seven years. In this article, I present the third major release of the Markdown package and the changes it brings compared to version 2.10.0. In the article, I target the three major stakeholders of the Markdown package. Writers will learn about the new elements which they can type in their Markdown documents, TEXnicians will learn how they can style Markdown documents in different TEX formats, and developers will learn about the governance and the development of the Markdown package and how they can extend Markdown with new elements. This article is a Czech translation of my talk at TUG 2023.

Ondřej Sojka, Petr Sojka, Jakub Máca, A roadmap for universal syllabic segmentation; pp. 125–138

An extended version of the article with the same title from *TUGboat* 44:2.

Barbara Beeton, Co by každý (LA)TEXový nováček měl znát [What every (LA)TEX newbie should know]; pp. 139–152

A Czech translation of the article from *TUGboat* 44:2. Translation by Jan Šustek.

Vít Starý Novotný, Sazba textu české lidové písně „Když jsem já sloužil" pomocí modulu l3seq jazyka expl3 [Typesetting the lyrics of the Czech folksong "Když jsem já sloužil"]; pp. 153–164

The language of TEX was developed for typesetting books. Although it is Turing-complete, it was not designed for software development. Whereas writing and designing documents is straightforward in plain TEX, programming is difficult due to a lack of basic data structures and complex macro expansion, both quite different from modern imperative programming languages.

In the LuaTEX engine, authors can also program in the imperative programming language Lua. Although Lua does not share the limitations of plain TEX, passing data between TEX and Lua is not straightforward and important information such as token category codes are lost in transit.

The expl3 programming language combines the best of both worlds and allows authors to program in TEX in a way that is similar to modern imperative programming languages.

In this article, I introduce the l3seq module of the expl3 language that provides the list data structure. Using l3seq, I typeset the lyrics of the Czech folksong *Když jsem já sloužil.* I also compare the l3seq implementation with one in plain TEX.

[Received from Vít Novotný.]

TUG financial statements for 2023

Karl Berry, TUG treasurer

The financial statements for 2023 have been reviewed by the TUG board but have not been audited. The totals may vary slightly due to rounding. As a US tax-exempt organization, TUG's annual information returns are publicly available on our web site, below.

Revenue (income) highlights

Membership dues revenue was slightly down in 2023 compared to 2022; we ended the year with 1,162 paid members, 12 fewer than in 2022; this is better than expected, since there were no DANTE joint members in 2023, a situation that is remedied for 2024.

The 2023 online conference had a small loss, due mostly to unfavorable exchange rate variations. General contributions and product sales returned to their normal levels after the one-time large revenue items in 2022. Thus, overall, 2023 income was down around 33%.

Other highlights; the bottom line

TUGboat production and mailing fees increased substantially. With that and one-time costs incurred with a change in the office, our bottom line for 2023 was negative: −$26,167.

Balance sheet highlights

TUG's end-of-year asset total decreased, following that loss.

Committed Funds are reserved for designated projects: LaTeX, CTAN, MacTeX, the TeX development fund, and others (`https://tug.org/donate`). TUG charges no overhead to administer these funds.

The Prepaid Member Income category is member dues that were paid in earlier years for the current year (and beyond). The 2023 portion of this liability was converted into regular Membership Dues in January of 2023. The payroll liabilities are for 2023 state and federal taxes due in January, 2024.

Notes for 2024

We have increased membership fees slightly in 2024, for the first time in many years, as inflation and shipping costs have not stood still. Worldwide support from members and donations are what allow us to continue, so thank you! As always, we welcome ideas for new TUG benefits or activities.

◇ Karl Berry, TUG treasurer
`https://tug.org/tax-exempt`

TUG 12/31/2023 (vs. 2022) Revenue, Expense

	Dec 31, 23	Dec 31, 22
ORDINARY INCOME/EXPENSE		
Income		
Membership Dues	75,918	76,940
Product Sales	3,655	20,008
Contributions Income	12,597	37,055
Annual Conference	(989)	4,325
Interest Income	4,144	742
Advertising Income	340	375
Reimbursed Expenses	(1,600)	375
Total Income	94,065	139,445
Cost of Goods Sold		
TUGboat Prod/Mailing	(29,540)	(22,639)
TUGboat Crossref	(490)	(369)
Software Prod/Mailing	(3,120)	(2,818)
Members Postage/Delivery	(2,480)	(1,822)
Lucida Sales to B&H	(1,495)	(9,595)
Member Renewal	(639)	(520)
Total COGS	(37,753)	(37,763)
Gross Profit	56,312	101,682
Expense		
Office Overhead	(15,783)	(12,647)
Payroll Expense	(60,496)	(71,565)
Professional Fees	(41)	
Interest Expense	(6)	
Total Expense	(76,776)	(84,212)
Net Ordinary Income	(20,464)	17,470
OTHER INCOME/EXPENSE		
Prior year adjustment	(5,703)	5,921
NET INCOME	(26,167)	23,391

TUG 12/31/2023 (vs. 2022) Balance Sheet

	Dec 31, 23	Dec 31, 22
ASSETS		
Current Assets		
Total Checking/Savings	168,572	198,499
Accounts Receivable	0	2,335
Total Current Assets	168,572	200,834
LIABILITIES & EQUITY		
Current Liabilities		
Committed Funds	51,538	53,524
Administrative Services		1,443
Prepaid Member Income	13,290	11,395
Payroll Liabilities	978	3,539
Total Current Liabilities	65,806	71,901
Equity		
Unrestricted	128,934	105,542
Net Income	(26,166)	23,392
Total Equity	102,768	128,934
TOTAL LIABILITIES & EQUITY	168,574	200,835

TEX conferences and lectures

BachoTEX 2024
Bachotek, Poland
May 1–5

`gust.org.pl/bachotex/2024-en`

TUG 2024
Prague, Czech Republic
July 19–21

`tug.org/tug2024`

GuiT 2024
Brescia, Italy
May 4

`guitex.org`

ConTEXt 2024
Lutten, The Netherlands
August 17-23

`meeting.contextgarden.net/2024`

GUTenberg
Exposés mensuels
(monthly presentations)

`gutenberg-asso.fr/`
`-Exposes-mensuels-`

SUN (NOT TO SCALE)
MERCURY
VENUS
EARTH
MARS
JUPITER
Optimistic aliens measure space typographically
1 AU
SPACE TIP: IF YOU'RE EVER LOST IN THE INNER SOLAR SYSTEM, YOU CAN JUST TYPE OUT THE PHRASE "OPTIMISTIC ALIENS MEASURE SPACE TYPOGRAPHICALLY" IN TIMES NEW ROMAN AND USE THE DOTS AS A MAP.

`xkcd.com/2863`

Calendar

2024

Apr 16 – 20	Association Typographique Internationale, ATypI Brisbane 2024, “Crafted Technology”, Brisbane, Australia. `atypi.org/conferences-events/atypi-brisbane-2024`
May 1 – 5	BachoTEX 2024, “Composed thoughts”, 29th BachoTEX Conference, Bachotek, Poland. `www.gust.org.pl/bachotex/2024-en`
May 4	GuIT Meeting 2024, 20th Annual Conference, Brescia, Italy. `www.guitex.org/home/en/meeting`
Jun 1	Type Paris: Now24, talks on typography’s visible impact; Workshops Jun 2: Workshop No. 19: Glyph Font Making 101; Workshop No. 20: Script lettering. Paris, France. `typeparis.com/now24`
Jun 4 – Jul 12	Type Paris Summer 24, intensive type design program, Paris, France. `typeparis.com`
Jun 6 – 8	XML Prague 2024, a conference on markup languages and data on the web. University of Economics, Prague, Czech Republic. `xmlprague.cz`
Jun 12 – 13	Year of Printing Heritage Conference, Centre for Printing History & Culture, CPHC, Birmingham, UK. `cphc.org.uk/events`
Jun 26 – 28	Twenty-second International Conference on New Directions in the Humanities, “Traveling Concepts: The Transfer of Ideas in the Humanities”, Sapienza University of Rome, Rome, Italy, and online. `thehumanities.com/2024-conference`
Jul 1 – 5	SHARP 2024, “Global Book Cultures: Materialities, Collaborations, Access”, Society for the History of Authorship, Reading & Publishing, University of Reading, Berkshire, UK. `sharpweb.org/main/conferences`

TUG 2024	**Prague, Czech Republic.**
Jul 19 – 21	The 44th annual meeting of the TEX Users Group. Presentations covering the TEX world `tug.org/tug2024`

Jul 24 – 27	TypeCon 2024, Revolution Hall, Portland, Oregon. `typecon.com`
Jul 28	**Final papers due for TUG 2024 proceedings.**
Jul 28 – Aug 1	SIGGRAPH 2024, Denver, Colorado. `s2024.siggraph.org`
Jul 29 – Aug 2	Balisage: The Markup Conference (virtual). `www.balisage.net`
Aug 6 – 9	Digital Humanities 2024, Alliance of Digital Humanities Organizations, “Reinvention & Responsibility”, Arlington, Virginia, and online. `dh2024.adho.org`
Aug 17 – 23	18th International ConTEXt Meeting, “Keeping up”, Lutten, The Netherlands. `meeting.contextgarden.net/2024`
Aug 20 – 23	24th ACM Symposium on Document Engineering, Adobe, San Jose, California. `doceng.org/doceng2024`
Oct 4	*TUGboat* **45**:3, submission deadline.
Oct 23 – 25	Grapholinguistics in the 21st century — From graphemes to knowledge, Università Ca’ Foscari, Venice, Italy. (Donald Knuth is listed as a presenter) `grafematik2024.sciencesconf.org`

Status as of 15 April 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.html`. Interested users can subscribe and/or post to the related mailing list, and are encouraged to do so.

Other calendars of typographic interest are linked from `tug.org/calendar.html`.

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 (at) lk.net`
Web: `www.borisv.lk.net`

TEX and LATEX consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom LATEX packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in TEX and three decades of experience in teaching & training. I have authored more than forty packages on CTAN as well as Perl packages on CPAN and R packages on CRAN, published papers in TEX-related journals, and conducted several workshops on TEX and related subjects. Among my customers have been 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 (at) 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.

Hendrickson, Amy
57 Longwood Avenue Apt. 8
Brookline, MA 02446
+1 617-738-8029
Email: `amyh (at) texnology.com`
Web: `www.texnology.com`

Full time LATEX consultant for more than 30 years; 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 `https://texnology.com/train.htm` for sample of course notes.

Call or send email: 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: `david.latchman (at) texnical-designs.com`
Web: `www.texnical-designs.com`

LaTeX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized LaTeX packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

LaTeX Typesetting
Auckland, New Zeland
Email: `enquiries (at) 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 (at) wardepub.com`
Web: `myprojectnotebook.com`

I have been in academic publishing for 30+ years. I was a linguistics major at Stanford in the mid-1970s, then started a publishing career. I knew about TeX from editors at Addison-Wesley who were using it to publish beautifully set math and computer science books.

Long story short, I started using LaTeX for exploratory projects (see website above). I have completed typesetting projects for several journal articles. I have also explored the use of multiple languages in documents extensively. I have a strong developmental editing background in STEM subjects. If you need assistance getting your manuscript set in TeX I can help. And if I cannot help I'll let you know right away.