TUGboat

Volume 44, Number 2 / 2023
TUG 2023 Conference Proceedings

TeX Users Group

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

Individual memberships

2023 dues for individual members are as follows:

- Trial rate for new members: $30.
- Regular members: $105.
- Special rate: $75.

The special rate is available to students, seniors, and citizens of countries with modest economies, as detailed on our web site. Members may also choose to receive *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 2023 is $115.

Institutional memberships

Institutional membership is primarily a means of showing continuing interest in and support for TeX and TUG. It also provides a discounted membership rate, site-wide electronic access, and other benefits. For further information, see `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.

[printing date: August 2023]

Printed in U.S.A.

Board of Directors

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`

TeXnical support, public mailing list:
`support@tug.org`

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

2023 Conference Proceedings

TeX Users Group
Forty-fourth annual TUG conference
Bonn, Germany
July 14–16, 2023

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
TUGboat Editor Barbara Beeton
Proceedings Editor Karl Berry

Volume 44, Number 2, 2023
Portland, Oregon, U.S.A.

TUG 2023 — Bonn, Germany — July 14–16, 2023

The forty-fourth annual conference of the TeX Users Group

https://tug.org/2023 ▪ tug2023@tug.org

Conference committee

Barbara Beeton
Karl Berry
Gert Fischer, co-principal organizer
Ulrike Fischer, co-principal organizer
Robin Laakso
Paulo Ney de Souza

Conference artwork: Jennifer Claudio

Sponsors

TeX Users Group
DANTE e.V.
Google
GUTenberg Association
Overleaf
Pearson Addison-Wesley
STM Document Engineering Pvt Ltd
with generous assistance from many individual contributors.

Thanks to all!

Participants

Nils Ackermann
Giedrius Andreikėnas, VTeX
Nelson H F Beebe
Barbara Beeton, TUGboat
Doris Behrendt, DANTE e.V.
Denis Bitouzé, LMPA, Univ. du Littoral Côte d'Opale
Johannes Braams, TeXniek
Erik Braun, CTAN
Gyöngyi Bujdosó, University of Debrecen
David Carlisle
Paulo Cereda, Overleaf
Ben Davies, Overleaf
Yann Denichou
Christine Detig, Schrod Net & Publication Consultance
Luzia S. Dietsche, DANTE e.V.
Karin Dornacher, DANTE e.V.
Carl-Clemens Ludwig Ebinger
Gert Fischer, Bär Backup Crew - B.B.C.
Ulrike Fischer, The LaTeX Project
Thomas Flinkow
Ben Frank, Island of TeX
Velicia "Vhe" Frazier-Ratajczak
Deimantas Galčius, VTeX
Sasha Göbbels, Overleaf
Ulrich Grabowsky
Steve Grathwohl
Enrico Gregorio, Università di Verona
Patrick Gundlach, speedata GmbH
Jochen Günther
Jasper Habicht
Jim Hefferon, St Michael's College
Tom Hejda, Overleaf
Klaus Höppner, DANTE e.V.
Jérémy Just, ENS Lyon, CNRS
Jonas Karneboge
Oliver Kopp, JabRef e.V.
Stefan Kottwitz, LaTeX.org
Reinhard Kotucha
Marcel Krüger
Robin Laakso, TeX Users Group
Eberhard W Lisse, Omadhina Internet Services Ltd
Manfred Lotz, CTAN, DANTE e.V.
Stephan Lukasczyk, DANTE e.V., University of Passau
Jakub Máca
Carla Maggi
Henri Menke, PGF/TikZ
Ralf Mispelhorn
Frank Mittelbach, The LaTeX Project

doi.org/10.47397/tb/44-2/tb137confinfo

Ross Moore, Macquarie University
Dennis Müller, FAU Erlangen-Nürnberg
Philipp Müller
Gerd Neugebauer, CTAN
Leah Neukirchen
Vít Novotný
Marei Peischl, peiTEX
Bernd Raichle
Henning Hraban Ramm
Thomas Ratajczak, German Armed Forces
Oliver Rath, genua GmbH
T Rishikesan, STM Document Engineering
Linus Romer
Martin Ruckert, Munich University of Applied Sciences
Victor Sannier, GUTenberg Association
Volker RW Schaa, DANTE e.V.
Thomas A. Schmitz, Bonn University
Michael Schlueter, Schlueter Consultancy
John Schrader
Joachim Schrod, Schrod Net & Publication Consultance
Torsten Schuetze
A M Shanmugam, STM Document Engineering
Martin Sievers, DANTE e.V.
Ondřej Sojka
Petr Sojka, Masaryk University
Jonathan P. Spratte
Mikael Persson Sundqvist, Lund University
Jan Šustek
Heiko Thimm, Kremer und Thimm GmbH
Sigitas Tolušis, VTEX
Rajeesh Keezhe Veettil
Didier Verna, EPITA Research Lab
Boris A Veytsman, Chan Zuckerberg Initiative, George Mason University, TUG
Ulrik Vieth
Stefan Vollmar, MPI for Metabolism Research
Eileen Wagner
Anselm von Wangenheim, pdf2tex.com
Alan Wetmore
Alexander Willand
Joseph Wright, The LaTeX Project
Uwe Ziegenhagen

TUG 2023 program

Thursday, July 13	14:00	*PDF Developers' Workshop*	
	15:00	*Excursion — Guided walking tour of downtown Bonn*	
	19:00	*Reception & registration at Hotel Leoninum*	
Friday, July 14	08:00	*registration*	
	08:30	Boris Veytsman, TEX Users Group	*Welcome*
	08:45	Ulrike & Gert Fischer, Carla Maggi, Paulo Cereda, samcarter	*Behind the scenes of the Great TikZlings Christmas Extravaganza*
	09:15	Oliver Kopp, JabRef e.V.	*JabRef as BibTEX-based literature management software*
	09:45	Jan Šustek	*On generating documented source code by blocks in TEX*
	10:15	Barbara Beeton, *TUGboat*	*What every LaTeX newbie should know*
	10:45	*break & registration*	
	11:15	Martin Ruckert, Munich University of Applied Sciences	*News from the HINT project*
	11:45	Dennis Müller, FAU Erlangen-Nürnberg	*An HTML/CSS schema for TEX whatsits*
	12:15	Patrick Gundlach, speedata GmbH	*News from boxes and glue: How do the TEX algorithms help in developing a new typesetting engine?*
	12:45	*lunch*	
	14:00	Joseph Wright, samcarter	*Beamer news*
	14:15	samcarter	*The* `tcolorbox` *inner beamer theme*
	14:30	Boris Veytsman, Chan Zuckerberg Initiative, George Mason Univ., TUG	*The update of the* `nostarch` *class*
	15:00	Ben Davies, Overleaf	*Bumpy road towards a good LaTeX visual editor*
	15:30	Didier Verna, EPITA Research Lab	*Interactive and real-time typesetting for demonstration and experimentation*
	16:00	*break*	
	16:30	Eberhard W. Lisse, Omadhina Internet Services Ltd	*Introduction to Typst*
	17:00	Jakub Máca, Petr Sojka, Ondřej Sojka	*Universal syllabic pattern generation*

TUG 2023 program (continued)

Saturday, July 15	08:30	Henning Hraban Ramm	*Architectural guides for Bonn — book production with ConTEXt*
	09:00	Thomas Schmitz, Bonn University	*Producing different forms of output from XML via ConTEXt*
	09:30	Vít Novotný	*Markdown 3: What's new, what's next?*
	10:00	Rishi T, Apu V, Hàn Thế Thành, Jan Vaněk, STM Document Engineering	*Primo — The new sustainable solution for publishing*
	10:30	*break*	
	11:00	Ross Moore, Macquarie University	*Tagged PDF, derived HTML and aspects of accessibility*
	11:30	Ulrike Fischer, LaTeX Project	*Automated tagging of LaTeX documents — what is possible today?*
	12:00	Joseph Wright, LaTeX Project	*Supporting backends in expl3*
	12:30	Frank Mittelbach, LaTeX Project	*The LaTeX Companion, 3rd edition — Anecdotes and lessons learned*
	13:00	*lunch*	
	14:00	Jim Hefferon, St Michael's College	*Using Asymptote like MetaPost*
	14:30	Linus Romer	*Curvature combs and harmonized paths in MetaPost*
	15:00	Rajeesh KV	*Metafont, MetaPost and a complex-script typeface*
	15:30	Victor Sannier, GUTenberg Association	*A METAFONT for rustic capitals*
	16:00	*break*	
	16:30	Ulrik Vieth	*An updated survey of OpenType math fonts*
	17:15	Mikael Sundqvist, Lund University	*Extending OpenType math, making choices*
	17:45	Tom Hejda, Overleaf	*TEX Live and Overleaf revisited*
Sunday, July 16	08:30	Island of TEX	*Living in containers — on TEX Live in a docker setting*
	09:00	Joseph Wright	*Further adventures in Unicode-land: Refining case changing*
	09:30	Oliver Kopp	*The LaTeX template generator: How micro-templates reduce template maintenance effort*
	10:00	Island of TEX	*The Island of TEX 2023 — sailing the smooth seas of ideas*
	10:30	*break*	
	11:00	Frank Mittelbach	*38 years with LaTeX — A personal picture story*
	12:00	Boris Veytsman	*Closing*
	≈ 12:15 pm	*end/lunch*	

Conference photo courtesy Alan Wetmore.

A rally in Bonn: TUG 2023

Eileen Wagner

Thursday, July 13

TUG 2023 took place in Bonn, Germany, after three years of online-only meetings due to the COVID-19 pandemic. Our local hosts, Ulrike and Gert Fischer, invited us to the Hotel Leoninum, a retirement home and hotel that provided a sense of calm and conviviality.

Beyond the usual updates on kernel, key packages, and favorite editors, this year's topics focused on continued efforts in accessibility, related research around hyphenation, and various experiments in Unicode and more. It wouldn't be TUG without the riveting demos, be it specific publishing house setups, private practice workflows, or yes, even writing *The LaTeX Companion* itself. There was a stream of live tweets and toots about the conference on `#TUG2023`.

Friday, July 14

The conference took place at the Old Church of the hotel, a repurposed seminary from the 19th century. This was not only befitting the sanctity of the TUG, but also enabled speakers to preach to the choir, from the quire.

The president, Boris Veytsman, opened the conference at 8:30 sharp. He noted that this is the first in-person conference since the beginning of the COVID-19 pandemic, which is special for him and many attendees.

Ulrike and Gert Fischer, Carla Maggi, Paulo Cereda and samcarter kicked off the talks with a lighthearted peek behind the scenes of generating the annual Great TikZlings Christmas Extravaganza video.[1] samcarter went into detail on how each scene was created in Beamer and then stitched together from PDF to PNG to MP4. The talk ended with a compilation of various duck-themed animations.

Next, Oliver Kopp urged the audience to (re)consider JabRef as BibTeX-based literature management software. Their recent work rewired the GUI and switched to BibTeX as its internal data model, which means no conversions are needed. New features such as integrated web search, drag-and-dropping PDFs with metadata, grouping options, and data quality control are truly exciting. Almost as importantly, Oliver described their efforts to have JabRef included as a potential student project at university computer science programs. This meant a lot more contributions for JabRef, and a lot more computer science students who worked on a real-world package with their code actually being used. Their contribution framework on GitHub[2] is a great place to start.

Jan Šustek followed with his work on generating documented sources by blocks. The goal is to write source code as well as its documentation in a single file. This was made possible by modifying certain OpTeX macros and using nested blocks. The audience was equally awed by the capability of this macro and the fact that they met a real-life plain TeX user — on macOS, no less.

Barbara Beeton reviewed frequently asked questions on `tex.stackexchange` so you don't have to! She turned these questions into an introduction to key concepts in TeX, titled "what every newbie should know". The article is highly recommended for those frequently introducing TeX to new users.

After a coffee break, Martin Ruckert came with news from the HINT project. HINT is a file format, viewer, and engine (HiTeX) that renders TeX documents in a resizable and searchable viewer. (See TUG 2019 and 2020 for more details.) He introduced the issue of optimizing for small font sizes (<10 pt). Outline fonts, along with some tricks in rounding and interpolation, can help glyphs become better defined. Support for links, labels, and outlines were added in HINT as well. Martin showed that some commands are better than others when supporting variable window sizes. For that, HINT needs to design more macros. Did you know that you need to type 'La TEX' in Unicode to find 'LaTeX' because between 'a' and 'T' we use a glue and not a kern? Martin's next projects might include bi-directional typesetting and subpixel rendering. Thankfully he's going into retirement from teaching this fall, so we can expect even more updates soon.

Dennis Müller then introduced an HTML/CSS Schema for TeX primitives. But first he talked about RusTeX, which is an engine in Rust that is close to pdfTeX. Right now there is a difficult tool chain for producing HTML involving LaTeXML, OMDDoc, MMT, and more, which "might only work on Martin Kohlhase's computer". Dennis' goal was to combine everything in one package with the same representation format for both humans and machines. This way you can inject additional services such as JavaScript. He used an AI — "a good old-fashioned symbolic AI" — to help with the conversion, and so far the results look promising.

Patrick Gundlach makes a lot of catalogues, and is maintaining a project called 'boxes and glue' that brings TeX's typesetting engine to more folks (implemented in Go). His motivation is mainly that there

[1] `github.com/TikZlings`

[2] `github.com/orgs/JabRef/projects/3`

doi.org/10.47397/tb/44-2/tb137wagner-tug23

are some trivial limitations for LuaTEX: no https requests, for example, which he really needs! He has his work cut out for him: on the backend with fonts, language, nodes, PDF library, and on the frontend with font families, colors, CSS/HTML, accessibility, page layout, and so on. (And then an application on top!) His boxes and glue engine is "pretty fast" outputting 300 pages per second, which is much better than the 50 pages per second he had with his old catalogue software. He also described some lessons learned, starting with "PDF looks like an innocent file format..." He confirmed the audience's suspicion that, once set up properly, the typesetting is much faster than the previous InDesign process.

With that, we headed to lunch, with tables sprawling from the hotel restaurant to the sunny terrace. Delegates gathered shortly before the afternoon sessions to take a group picture. One of the sponsors distributed a number of rubber ducks, brightening the post-lunch spirits.

samcarter (and Joseph Wright) gave updates on `beamer`. While their efforts still focused on maintaining backwards compatibility, they also made sure to ship some usability improvements. For example, transparent backgrounds now work as expected; there is a flexible interpretation of the `aspectratio` option; title pages are more modular; the geometry calculation was renewed.

This transitioned to another talk by samcarter on `tcolorbox`, a new theme for the `beamer` class. It replaces normal beamer blocks with `tcolorbox`es of the same look and feel. `tcolorbox` makes it easy to modify the appearance of blocks, accommodating all kinds of user requests ("Can I have the rectangular block with rounded shadows?"). Keep them coming!

Boris Veytsman took the floor to describe his `nostarch` class. No Starch Press is a publishing house that switched their tooling from InDesign to LaTEX, very much thanks to Overleaf's wider adoption. The TEXnical problems he faces there can be unusual. For example, url splitting on hyphens is ambiguous between hyphen (linebreak) or hyphen (url). No Starch has a convention to split *before* hyphens. Another example: captions should be the same width as the corresponding figure. Boris insisted that he can't fix a bug with `amsmath` (Bad mathchar 32768), but despite his claim that "the magic of category codes is sometimes black magic", a delegate, David Carlisle, found the issue (including line reference) by the end of the presentation, and the fix is in the current release.

Overleaf's Ben Davies followed with a presentation of their new Visual Editor. This development comes from well-known issues: many people — collaborators or proofreaders or just "young people who are intimidated by any code" — might not be familiar with LaTEX, so Overleaf wanted to offer a way for them to contribute without using TEX syntax. This was made possible by the switch from CodeMirror5 to CodeMirror6. The new Visual Editor is a WYSIWYG editor that hides code and adds previews of maths and figures. Track changes and comments work with both the Code Editor and Visual Editor. Ben discussed many user experience decisions, such as introducing a visual "undo" button that gives people the confidence to make mistakes and experiment. The crucial issue is to balance how much code to hide, and refraining from too much WYSIWYG conditioning which can feel limiting. So far, 2% of Overleaf users are making use of the Visual Editor.

The visual theme was continued by the next speaker, Didier Verna, who demonstrated his visualisation software for interactive and real-time typesetting. TEX engines are production systems that are not meant for playing around, and so his tool focuses on experimentation and demonstration instead. His program ETAP[3] showcases kerning, ligatures, baselines, etc., in typesetting algorithms. (It of course includes the Knuth-Plass algorithm.) After a brief detour on programming in multi-paradigm languages that allow inheritance and polymorphism, he went on to demonstrate the power of ETAP to produce statistical reports on the behavior of the various algorithms. He showed a number of benchmarking tests for efficiencies of various algorithms. He hopes to add more parameters in the future — `microtype` and a tolerance threshold (sloppiness) are clear candidates.

Eberhard Lisse took on a review of Typst, a recent addition to the typesetting ecosystem. Eberhard started using LaTEX early on and, like so many of us, never wanted anything else. He gave an in-depth view into the use of TEX in his medical practice, and asked "Is TEX ready for prime time?" Even though his staff has begrudgingly learned LaTEX and he's gotten very far with customising his tool chain, his answer is a sad "no". "For casual use it's much too complicated," he concluded. More reason, then, to be excited about a new alternative: Typst. Typst is both a typesetting engine, a collaborative editor, and a platform built by two computer science students from Berlin, who may have shared similar feelings about the production-readiness of LaTEX. In Eberhard's view, Typst fills the gap between advanced tools (TEX) and simpler tools (Word/Google Docs): it is highly capable, blazingly fast, and uses a simple markup language. It is also less powerful than

[3] `github.com/didierverna/etap`

TEX. He proceeded to point to missing features — namely floats, indices, interactions with the environment, and labels — but remains hopeful that sensible decisions will be made. *Eberhard is also going into retirement this fall, so we can expect more experiments in Typst!*

Next, a research group from Masaryk University consisting of Ondřej Sojka, Jakub Máca, and Petr Sojka presented a roadmap for universal syllabic segmentation. The tagline: give patterns a chance! They are studying the similarity of languages when it comes to hyphenation patterns, starting with Czech and Slovak, and following up to nine languages (cz, sk, ka, el, pl, ru, tr, tk, ua). Their results: syllabic segmentation across languages is possible using Judy arrays; and their hypothesis: universal patterns are feasible, too, with high impact on virtually all typesetting engines.

The conversation then seamlessly transitioned to the TUG Annual General Meeting (see separate report in this issue). Delegates left the hotel in time to eat out on this warm summer evening.

Saturday, July 15

Henning Hraban Ramm showcased the architectural guides for Bonn that his publishing house[4] made, using first InDesign and then ConTEXt in production. Some of his customizations in ConTEXt include OpenStreetMap integrations, beautiful bleed and trim boxes, and shadow captions for white text on a monochrome background. When asked how long it took to produce the book, he said that the basic typesetting was very quick, but it took hours to set it up; in InDesign it was the other way around.

The morning continued with beautiful examples from Thomas Schmitz, who teaches ancient Greek at the University of Bonn. He switched to ConTEXt years ago with the ambitious goal of having all lecture content and notes in the same source document. For him this meant XML, with its advantage of easy code reuse and outputs in different formats. Examples of these formats are lecture slides, course schedules, translations (philology), printout for students, bibliography for conferences. This amounted to over 30k lines over one term. Thomas demonstrated his setup with an integrated Lua interpreter. His presentation module is available on the ConTEXt Garden.[5] Upon the question of any preprocessing setups (linting and checks), he responded that that won't be possible because of his editor (Emacs) — cheers in the room.

Vít Novotný came next with a Markdown 3 update. He observed that LATEX is easy to write, but hard to read! This presents a challenge, since most of the time writers need to read what they are writing. Markdown is not just a great option for those wanting a non-distracting syntax; it is also a great option for TEXnicians who need to work with clients and publishers who do not use TEX syntax. Vít went on to discuss the Markdown syntax of choice, CommonMark by Jeff Atwood and John MacFarlane.[6] Tables, footnotes, and citations are not included though, and for structured metadata, a YAML file is still required. The exciting feature is the "hybrid" mode that allows authors to switch back and forth between Markdown and TEX syntax. Until now, Markdown-enabled TEX files were not interoperable with other Markdown clients. Markdown 3's hybrid syntax does not break interoperability. It also introduces some neat features like tables, task lists, superscripts, and more. It added TEX4ht for websites, OpTEX support, and also updated 'pandoc-to-markdown'; `lua-tinyyaml` is now on CTAN. Perhaps more importantly, Markdown 3 is now en route to have better governance and community management, adding rooms in Discord and Matrix to welcome developers and *authors* alike. The stable release is coming soon. The session ended with an admission of guilt: Vít created his presentation with Google Slides.

Next, Rishi T from STM Document Engineering announced a new editor, Primo. As a company that works closely with publishers, they have built many tools that ease the authoring-to-publishing workflow. With Primo, they wanted to create an all-in-one solution for many known issues in the submission process. These issues include: authors can't understand journal's requirements, authors cannot collaborate easily, missing material during submission, the back and forth querying, and general technical constraints in the submission systems. Primo is a cloud-based authoring, submission, proofing framework that hopes to be self-explanatory and user-friendly. It will offer a WYSIWYG and non-WYSIWYG mode as well as collaboration, which the audience saw in screenshots. It is XML-based and a DTD-compliant tool. There was great disappointment in the room when the timeline was announced: the platform will be released between April 2024 and April 2025 in three phases.

After the coffee break, Ross Moore gave an overview of accessibility principles and practices via a pre-prepared video. He gave a thorough tour of TEXed documents in various PDF readers, evaluating

[4] `www.dreiviertelhaus.de`

[5] `modules.contextgarden.net/cgi-bin/module.cgi/ruid=5038752680/action=view/id=31`

[6] `commonmark.org`

them against the PDF/UA standard[7] set by the W3C and WCAG. The display of bookmarks vs. tables of contents vs. hyperlinks is not always compatible with navigation, for example, and a lot of information is hidden in fields that are not read by screen readers (hover-over). The best practice is to include alt-descriptions for information blocks, and adding navigation shortcuts in sensible places. Ross also referenced best practices in accessibility design in web pages, which have more established standards and tools for evaluation. A well-tagged document can have its semantics carried over to web pages as well.

Following the general introduction to PDF accessibility, Ulrike Fischer gave an update on the `tagpdf` project. The goal is to add structure to PDFs to improve accessibility and reuse of data. Adding tags (alt-descriptions and other useful metadata) is already possible in the current release, but the challenge for this project is to make tagging automatic and easy to use. Some first problems (again echoing Ross's talk): free-of-charge PDF viewers don't show tags in the visual interface ("If people can't see it in their resulting PDF, they often don't want it!"); this also makes testing difficult. Moreover, many PDF viewers don't support PDF 2.0, which is important for proper tagging. Ulrike described tagging standard "Leslie Lamport documents" as tests — this is possible now as of the summer 2023 release! Some things are not yet supported, notably footnotes, sectioning in `memoir`, etc. More work ahead!

Tagging is not the only core construction happening now. Joseph Wright gave an update on `expl3`, very much motivated by the question "what does TEX NOT do out of the box?" TEX is "blind" to core document elements such as color and graphics — the boxes to be glued, so to say. Joseph took a closer look at these backend support files, such as `drivers.dtx` which includes a lot of clutter accommodating different developments of backends over the years. `expl3` tries to create a single source for `.def` files that would be compatible with core backends (pdfTEX, etc.). This principle applies not only to colors and graphics, but also hyperlinks, PDF constructs, tagging of course, drawings, and box transformations. As with most rewrites, the difficulty lies in finding the right level of abstraction. Given the immense undertaking, the audience wondered if it would be easier to create a specification or a set of primitives. This way, engine developers can carry the burden of implementing it respectively.

More news from the LATEX Project: Frank Mittelbach was ready to present *The LATEX Companion*, third edition. The "monster" weighs 3.5 kg and took five years to complete. A primary goal was to classify CTAN packages according to their functionality, usability, and correctness — and how they relate to other packages. After an initial review, Frank decided to focus on 500 (about 10% of all packages). For those who might compile a 1600-page book with over 100 fonts in the future, Frank shared his lessons learned:

- Test all examples included — better even to have them in production.
- Take care of pagination *after* you copyedit and finalise font sizes.
- Keep layout code as separate as possible, so you can distinguish between things you changed for the layout and things you changed "to make the page look right".
- Never underestimate the power of automated checking.

Frank welcomes scrutiny and contributions of course. Now off to lunch!

In another remote presentation, Jim Hefferon made a case for using Asymptote,[8] a descriptive vector graphics language for technical drawings. It is in part based on Metafont and MetaPost, but extends it from 2D to 3D. Asymptote uses a single source file for related graphics, meaning (unlike Ti*k*Z) all graphics are *outside of the document*. Jim showcased a number of Asymptote examples.

Linus Romer, in designing fonts, threw himself in at the deep end of curvature combs. Most font editors offer curvature-related tools, and curvature combs are one of them. He described his implementation of curvature combs in MetaPost, along with a number of harmonization algorithms that helped smooth out the paths. As expected, there is lots of math behind curvy characters!

More font adventures: Rajeesh KV talked about shape-shifting Indic scripts. In Malayalam, any consonant followed by the vowel sign of u, ū, or ṛ are represented by a cursive consonant-vowel ligature. The glyph of each consonant has its own way of ligating with these vowel signs. We may call this "ligatures on steroids". Rajeesh developed a reusable component-based design for these Malayalam fonts using Metafont/MetaPost to assemble the characters. This shifts the paradigm from visual tools to code-based tools. His assembly line: MetaPost, SVG, FontForge, scripts, and finally OTF/TTF/WOFF2. It is even possible to specify width and angle of the pen for the shape library!

Victor Sannier representing the French TUG (Le Groupe francophone des Utilisateurs de TEX,

[7] `www.iso.org/standard/64599.html`

[8] `asymptote.sourceforge.io`

GUT) showcased his Metafont for rustic capitals. Rustic capitals as a type started in the 1st century, and had been regularized in the 4th and 5th century. He measured and analyzed many of them until eventually tracing them with a `draw_serif` macro he devised. He said that he would continue to design more rustic characters and welcomes feedback.

Meanwhile, Bonn is getting warm and humid, and the organizers have opened the doors and turned on the fans. Delegates opened bottles of sparkling water and cleverly mixed them with apple juice to create the inimitable *Apfelschorle*.

Another font talk followed. Ulrik Vieth had the unenviable task of reviewing all OpenType math fonts — so you don't have to. He both evaluated their completeness and design choice. He started with a history of OpenType math fonts which were first ready for use around 2010. The list is now over 20 official fonts, and more than 30 if you count not yet released or not properly licensed fonts. Some fonts have around 500 symbols, some have 1200. This largely depends on the availability of additional series such as script, fraktur, or blackboard bold.

Building on the previous talk, Mikael Sundqvist dug deeper into OpenType math. The root of many issues here is that there is no good standard! While Microsoft offers some instructions, it's unclear what should be included in a math font. Several OpenType math fonts were created by converting and extending older TeX fonts. Together with Hans Hagen, Mikael works on math in ConTeXt. He observed many frustrating inconsistencies: italic corrections are transformed to corner kerns, staircase kerns are inconsistent, accents from different fonts look completely different. Depending on the engine, the glyphs are also differently rendered. Extensibles and rules often don't work. Their efforts have corrected for many of these issues in ConTeXt. Moreover, his observations often feed back to the font designer, many of them responsive. He promised to publish an article outlining best practices for math font design.

For the final talk of the day, Tom Hejda proposed closer collaboration between TeX Live development and Overleaf. Tom reiterated the Overleaf view that TeX Live should be "all you need to use LaTeX". Due to their uptime requirements, Overleaf can only afford an annual deployment in Q3 every year; that version of TeX Live is then locked for a year for their users — roughly two million at this point. Their schedule, along with the pressure to have a version of TeX Live that's compatible with the Overleaf ecosystem (most importantly: the templates that are offered), prompted Tom to start a discussion on whether and how TeX Live developers and Overleaf can work together. This would also mean TeX Live could benefit from early deployments and extensive testing. The group recommended that Overleaf contact the core team as well as package maintainers as soon as possible when deployments fail. (It turns out that, just like Overleaf, the community is also not thrilled about hard deadlines!) It is also possible for Overleaf to test the early (beta) version of the newest TeX Live release. It was noted that more frequent deployments (continuous integration) would directly benefit their users.

With that, the group moved on to a banquet at *Konrad*'s on top of the Marriott Hotel. We spent much time on the terrace overlooking the Rhine, enjoying the view of the Seven Hills and the old parliament of West Germany. A three-course meal followed. Some delegates chose to walk back to the city center along the river, a calm and inspiring end to Day 2.

Sunday, July 16

For those quick to recover from the banquet, Day 3 began with Ben Frank's presentation on Docker containers,[9] a community service offered by the Island of TeX. The brilliant idea behind Docker: it is self-contained, meaning each TeX Live image ships with literally everything required to run it. Ben went on to describe two use cases that benefit from Docker. First, a small German maths journal that needs to be able to run older versions of TeX for historical issues. Second, student representatives who compile their weekly meeting minutes with a completely CI-based and ConTeXt-heavy setup. Future directions for dockerized TeX could be including more operating systems (such as Raspberry Pis) and layer-friendliness (only pulling changes). He added a public service announcement: do not use 'latest', it is not image-friendly.

Joseph Wright took the audience on further adventures in Unicode-land. The code for the Unicode case changing algorithm is now in the kernel. So far, the project largely involved updating case mappings for Unicode engines, automatic locale switching via `\BCPdata`, and full Unicode support for pdfTeX. Joseph showed more recent work and challenges. One key take-away: Greek, with all its cases and accents and exceptions, is a great language to stress test your Unicode engine! Further improvements are coming, of course, such as true titlecasing and a closer look at graphemes (human-perceived characters).

Oliver Kopp introduced his library of LaTeX templates[10] motivated by providing a simple IEEE

[9] `islandoftex.gitlab.io/community/projects/docker`
[10] `latextemplates.github.io`

template that works out of the box. Many existing templates do not include key packages correctly, such as `microtype` or `hyperref`, and don't include minimal examples as guidance. His templates contain a bit of templating language, and the tool includes a CLI for basic design choices. The setup via "microtemplates" (modules, so to say) can reduce the overhead of maintaining LaTeX templates. Importantly, there are automated CI checks to ensure that the templates are always compatible with the newest stable release.

Paulo Cereda, Overleaf community coordinator and Island of TeX core member, gave a rundown of the latest developments on the Island (there are many!). First off, `albatross`, a CLI that helps you find a font based on particular glyphs, was redesigned with border styles, font lookup, and graphemes. Users can use glyph, hexadecimal, or multiset union to search. The Island's community website was also updated, with an index pulling from all the READMEs of the projects. There are exciting ideas for the future, such as providing binary code for packages. The Islanders will also continue their quest to improve user experience. Finally, the Island is ready for a new visual identity. Help on all of the above is explicitly wanted: "visas to the Island are easy to attain!"

To close off TUG 2023, Frank Mittelbach shared the highlights of his last 38 years with LaTeX through a personal picture story with a mere 90 slides. It started with an invitation in 1989 to attend TUG at Stanford, and appropriately ended with finishing the new *LaTeX Companion* in 2023, documenting many friends, ideas, and past TUG convenings across multiple continents.

Brushing aside the sentimentalities, Boris (now outgoing president) shared that he was glad that the group was able to bring back TUG in full glory. He thanked the organizers for the great line-up and cultural program — which would be continuing onwards to an outing to the Seven Hills (Drachenfels and Königswinter). The group lingered on the terrace after lunch, bidding their farewells and exchanging last comments. Until next year!

Acknowledgment

I'd like to thank the TUG bursary for funding to support me in attending this conference.

◇ Eileen Wagner
Berlin, Germany
`hello (at) bumble dot blue`

TUG 2023 Annual General Meeting notes

Notes recorded by Karl Berry and Robin Laakso

Boris Veytsman, outgoing TUG president, opened the meeting in Bonn, Germany, at approximately 17:45 CEST. He gave a short greeting from Arthur Rosendahl, incoming TUG president, who unfortunately could not be present.

Klaus Höppner, TUG secretary, gave a TUG status update and financial report. He showed a series of slides, most of which are included in this report (slides are omitted here if they merely duplicate information from web pages):

1. The TUG board of directors. (`tug.org/board`)
2. "Formalities": Klaus reported on the results of the 2023 TUG election. He welcomed the incoming president, Arthur Rosendahl, and incoming board members, Max Chernoff, Tom Hejda, Jérémy Just, and Boris Veytsman; Boris stood for election to the board after stepping down as president). Klaus thanked outgoing board member Paulo Cereda and Boris for his serving six years as president. (`tug.org/election`)
3. "Members end of May 2022": Klaus noted that DANTE joint memberships were not continued this year, due to organizational issues at DANTE. Thus TUG membership may be affected.
4. "Profit & Loss 2022" showed the major income and expense categories in 2022. Klaus noted that there were three major donations in 2022 which are not expected to recur, so the total contributions in 2023 will likely be substantially less. (`tug.org/tax-exempt`)
5. "Assets and Liabilities" and "Committed Funds" were next, both as of the end of 2022. Klaus noted the funds with particularly significant balances: CTAN, LaTeX3, PDF Accessibility.
6. "International Conferences": Klaus reported on past and upcoming conferences, particularly noting the welcome return of BachoTeX a couple months ago. (`tug.org/meetings`)
7. "TeX Live/TeX Collection": Klaus reported that the DVD has been manufactured in Germany, with cooperation and support from many people. The total run was about 3300 discs. The delivery for TUG is en route. (`tug.org/texcollection`)
8. "Board Motions" (`tug.org/board/motions.html`):
   2022.4 TUG'23 in Bonn (unanimous).
   2022.5 Approval of the 2023 budget (passed 12–0; no response from 1 board member).

TUG 2023 Annual General Meeting notes
doi.org/10.47397/tb/44-2/tb137tug23-agm

2023.1 Support for Ukrainian students at BachoTEX (unanimous).

2023.2 Additional support for Ukrainian students at BachoTEX (unanimous).

There were no questions or discussion. The meeting was adjourned at 18:10 CEST.

Annual General Meeting 2023 of the TeX Users Group

Klaus Höppner (secretary) for the board

July 14, 2023

Members end of May 2022

End of May we had 1,060 paid members, with:

- 946 renewals, 43 new (26 of them trial, 6 joint)
- −28 compared to June 2021
- 90 institutional (+5), 62 joint members
- **Remark:** no joint membership with DANTE this year!
- 383 with electronic-only option
- 349 with auto-renewal option
- 31 of last year's 50 trial members renewed so far
- final numbers of last years:
  - December 2022: 1,162
  - December 2021: 1,210
  - December 2020: 1,189
  - December 2019: 1,238
  - December 2018: 1,214
  - December 2017: 1,178

Profit & Loss 2022

Income		Expenses		
Membership dues	76,940	Cost of goods sold		
Product sales	20,008		TUGboat	23,008
Contributions	37,055		Software	2,818
Annual Conference	4,325		Fonts	9,595
Other	1,117		Postage	1,822
			Other	520
		Office		
			Payroll	71,565
			Overhead	12,647
Sum	139,445	Sum		121,975
		Net ordinary income		17,470

Assets and Liabilities (status end of 2022)

Assets		Liabilities	
Checkings/Savings	198,499	Committed funds	53,524
Accounts Receivable	2,335	Admin services	1,443
		Conference	2,000
		Member income	11,395
		Payroll	3,539
Sum	200,834	Sum	71,901
		Equity	128,934

Committed Funds (status end of 2022)

Fund		Amount
Bursary		4,560
CTAN		9,792
GUST e-foundry		536
LATEX3		14,115
LuaTEX		1,816
LyX		70
MacTEX		6,561
PDF Accessibility		11,071
TEX Development		5,003
owed:	1,500	
available:	3,503	
Sum		53,524

International Conferences

Past

- ConTEXt meeting (Germany, Sept. 12–18, 2022)
- DANTE (online, Nov. 19, 2022)
- Journée GUTenberg (online, Dec. 12, 2022)
- BachoTEX (Poland, Apr. 29–May 3, 2023)
- GuIT meeting (Italy, May 20, 2023)

Upcoming

- ConTEXt meeting (Czech Rep., Sept. 10–16, 2023)

TEX Live/TEX Collection

- TEX Live 2023 released as planned
- Team: Karl, Norbert, Siep Kroonenberg, Akira Kakuto et al.
- TEX Collection DVDs produced by DANTE in Germany, in cooperation with TUG and various user groups, containing:
  - TEX Live
  - MiKTEX
  - MacTEX (Richard Koch)
  - CTAN snapshot (Manfred Lotz)
- Former proTEXt distribution for Windows is abandoned, replaced by a special MiKTEX subset defined by Klaus (due to space restrictions). Thanks to Christian Schenk for support!

What every (LA)TEX newbie should know

Barbara Beeton

Abstract

LATEX has a reputation for producing excellent results, but at the cost of a steep learning curve. That's true, but by understanding a few basic principles, and learning how to avoid some techniques that may seem obvious but often lead one into the weeds, it's possible to avoid some of that pain. Our goal here is to encourage good habits before bad habits have had a chance to develop.

Introduction

The examples presented here are drawn from two main sources.

- In the author's years as part of the TEXnical support team for a major math publisher, responsibilities included fielding questions from authors and writing user documentation.
- The online TEX forum at StackExchange[1] has provided a surfeit of questions both basic and advanced. A community effort has collected a list of "Often referenced questions", by topic, at `tex.meta.stackexchange.com/q/2419`.

Exhortation: Read the documentation. (This will be repeated.)

Vocabulary

There are several concepts that seem to be either missing from a new user's bag of tricks, or not clearly understood. Let's get them out of the way up front.

Template Many new (LA)TEX users think that the document class is the template for a particular style or publication. Not so, although the thought is going in the right direction. A template is a source (`.tex`) file that is an outline. It begins with `\documentclass` and contains a minimum of basic structural commands into which additional definitions and text can be inserted as appropriate. Ideally, the template itself can be compiled with no errors resulting, but without producing any useful output.

Command line Most new users these days enter (LA)TEX from an editor or other GUI, and launch a non-interactive job that will blithely keep on processing the file until it finishes (with or without errors) or hangs in a loop. Launching the compilation from the command line, on the other hand, allows one to interact with the session and, in certain cases, make corrections "on the fly", or if that's not possible, halt the job in case of an error before the collection of reported errors becomes unhelpful. One type of "fixable" error is a misspelled command:

```
! Undefined control sequence.
l.37 \scetion
             {Section}
?
```

Respond to this with the correct spelling;

```
i\section
```

hit "return", and continue; don't forget to correct the file when you come to a good stopping point.

A misspelled environment name can't be corrected this way; if that happens, cancel the job with an `x`, fix the file, and start over. Continuing a run after an unfixable error will just result in more error messages, most of which are meaningless and confusing, so it's best to avoid them.

Log file Every time a TEX job runs, it will create a log file. Learn where to find this file! In addition to errors and warnings, it will report all files that were read in, including version numbers for document class files and packages, pages processed, and, at the end, resources used. Only a few relevant items will be mentioned here, but in a paper based on an earlier talk [1], instructions are given for how to undertake serious debugging.

Conventions

In order to avoid overfull lines, error and warning messages shown here may be broken to fit the narrow columns of the *TUGboat* style. Many error messages output by LATEX will consist of several lines, the first being the message, and the next showing the number of the line on which the error is identified along with the content of that line, up through the error text. A following line, indented so that it, with the numbered line, completes the line as it appears in the input.

Although we will deal here mostly with details, please remember that the basic concept of LATEX is to separate content from structure.

Basic structure: Commands, modes, and scope

Here we deal with some fundamentals of LATEX.

Commands Instructions are given to (LA)TEX by means of commands, or "control sequences", which by default begin with a backslash (\). There are two varieties: those which consist of the backslash followed by one non-letter character ("control symbol"), and those of one or more letters ("control words") in which only letters (upper- or lowercase A–Z) are permitted (no digits or special characters). A control word may

[1] `tex.stackexchange.com`

doi.org/10.47397/tb/44-2/tb137beeton-basic

have one or more arguments (`\title{...}`) or stand by itself (`\alpha`). A "standalone" control word will be terminated by a space or any other non-letter. But a space after a control symbol will appear as a space in the output. Several control symbols are predefined to produce their own character in the output: `\#`, `\%`, `\$`, `\&`. For example, `\$` produces '$'.

A user can define new commands, or assign new meanings to existing commands. LaTeX provides `\newcommand` to create a brand-new definition. `\newcommand` checks to make sure that the command name hasn't been used before, and complains if it has. (The basic TeX `\def` *does not*.) If it's necessary to redefine a command that already exists, the recommended way is to use `\renewcommand` — but be sure you know what you're doing. For example, redefining `\par` is chancy, as LaTeX uses this "under the covers" for many different formatting adjustments, and it's very easy to mess things up.

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$\ddot{\sqrt{-1}}$m

Single-digit commands (`\0`, `\1`, etc.) are not predefined in core LaTeX, so are available for ad hoc use.

Environments An *environment* is a block of material between

`\begin{`⟨*env-name*⟩`} ... \end{`⟨*env-name*⟩`}`.

The environment name must match at beginning and end; if it doesn't, this error is reported in the log file and on the terminal:

```
! LaTeX Error: \begin{xxx} on input line nn
                ended by \end{yyy}
```

Most environments can be nested, but the proper sequence must be maintained.

Other commands are available to provide new definitions — `\NewDocumentCommand`, `\NewEnvironment`, `\NewDocumentEnvironment` and similar ones for redefinitions. For details on these, consult a current reference.

Modes Generally speaking, the current mode identifies where you are on the (output) page, but here we will take a point of view based on the input/source file.

There are three modes: vertical, horizontal and math.

Starting after `\documentclass` or after a blank line or an explicit `\par`, LaTeX is in vertical mode. Certain operations are best launched in vertical mode; more about this later.

Starting to input ordinary text is one way to enter horizontal mode. Other transitions from vertical to horizontal mode are `\indent`, `\noindent` and `\leavevmode`. Within horizontal mode, multiple consecutive spaces are treated as a single space; *consecutive* is essential here. An end-of-line (EOL) is treated as a space, even though it's not explicitly visible in the source file; a GUI that wraps lines may or may not (usually not) insert an EOL, and different operating systems define an EOL differently, but such differences are taken care of by the TeX engine. Spaces at the beginning of a line are ignored. More about spaces later on.

The third mode, math, can be embedded in-line in text or set as display material in vertical mode. Inline math is wrapped in `$` signs or surrounded by `\(...\)`. An unnumbered one-line display can be indicated by `\[...\]`. Multi-line math displays are best entered using the environments provided by the amsmath and mathtools packages. (Refer to the user documentation. mathtools loads amsmath, so it's not necessary to load both.) A math display is usually a continuation of the preceding paragraph, so don't leave a blank line between a display and the preceding text; to do so can result in an unwanted page break.

Within math mode, blank lines are not allowed; this was a decision made by Knuth, to catch unintentional input lapses, since math never continues across a paragraph break.

Scope Along with modes, there is the concept of scope, making it possible to localize definitions and operations.

Math mode is one instance of scope; certain characters and operations are valid only within math, and others are invalid there. Within text, math usually begins and ends with `$`, and these must be matched. Display math breaks the flow of text; closing a display returns to text mode unless followed by a blank line or `\par`. More about math later.

Another way of delimiting scope is to wrap the material in braces: `{...}`. Within this scope, the meaning of a command may be changed for temporary effect; the definition in effect before the opening brace will be restored as soon as the closing brace is digested. Instead of a brace pair, the commands `\begingroup`...`\endgroup` have the same effect.

Another way to have a scoped environment is to pack the material in a "box". This may be a minipage, \mbox or \parbox. Other boxes are defined in packages like tcolorbox.

Some environments (not all) are defined to be a scope. One such is the theorem environment, inside which text is italic; when the theorem ends, the text style automatically reverts to the document default.

Spacing in text

A goal of high-quality typesetting is even spacing in text. This is really possible only with ragged-right setting, where spaces are "natural width". But even margins are usually preferred, so TEX is designed to optimize spacing in that context.

In U.S. documents, spaces that end sentences are wider than interword spaces. This is not true for documents in other languages, and can be turned off with \frenchspacing. But in academic documents, frequent abbreviations can make it difficult to tell where sentences end. To avoid a too-wide space after an abbreviation, follow it by "\␣" (backslash-space):

`abc vs. xyz` (abc vs. xyz) vs.
`abc vs.\ xyz` (abc vs. xyz)

If the line shouldn't break after the abbreviation, follow the period by ~: `seen on p.~23.` (seen on p. 23.)

A similar, but reverse, situation can occur when an uppercase letter is followed by a period. This is assumed to be the initial of a name; it usually is, and an ordinary interword space is set. But sometimes the uppercase letter is at the end of an acronym, and that ends a sentence. In such a case, add \@ before the period, and it will restore the wider end-of-sentence space.

All this boils down to a simple rule: Except at the end of a sentence (and to a lesser extent after other punctuation symbols or within math), all spaces within the same line should be the same width. If they're not, something is fishy.

Spurious spaces Multiple spaces can infiltrate a source file in several ways, but the overwhelming majority are the result of trying too hard to define commands in such a way that they are visually pleasing (and easily readable). For example:

```
\newcommand{\abc}{
   \emph{abc def}
}
```

With this, the input "`word \abc\ word`" results in "word *abc def* word" with extra spaces inserted by our \abc command. The offending spaces can be evicted by inserting % where it will "hide" an EOL:

```
\newcommand{\abc}{%
   \emph{abc def}%
}
```

to produce the desirable "word *abc def* word". The % character starts a comment, i.e., ignores the rest of the input line, including the EOL.

Another source of extra spaces in the output can be caused by the presence of multiple consecutive elements that aren't part of the main text, like footnotes or index entries:

```
An important topic\index{abc}
\index{def}
\index{xyz}
is indexed several ways.
```

An important topic is indexed several ways.

Here, the EOL effect has again occurred (after "topic"), and these spaces are no longer contiguous. Again the % comes to the rescue:

```
An important topic\index{abc}%
\index{def}%
\index{xyz}
is indexed several ways.
```

An important topic is indexed several ways.

Do remember to leave *one* space.

Sometimes using a % is a bad idea Remember that a space terminates a control word and it's then discarded; that's one place where it's not necessary to input a %. But there are places where adding a % can really cause trouble.

After defining any numeric value, TEX will keep looking for anything else that can be interpreted as numeric, so if a line ends with \xyz=123, no % should be added. Or, if setting a rubber length (glue), say \parskip=2pc, TEX will keep looking for plus or minus; a better "stopper" is an empty token, {}. (If "plus" or "minus" is there and happens to be actual text, a confusing error message will be produced, but that is rare, and beyond the scope of this discussion.)

Really unexpected extra spaces Other possibilities exist that aren't so predictable. Here's one that was the subject of an online question. A text with \usepackage{colorbox} (it can also happen with tcolorbox) had a colorized letter surrounded by spaces in the middle of a word. Oo p s! A small frame was applied around the colored element by the package:

```
\usepackage{colorbox}
\newcommand{\pink}[1]{%
  \colorbox{red!20}{#1}}
Oo\pink{p}s!
```

Oo p s!

Explicitly omitting the buffer inside the frame was the solution provided by the package documentation:

```
\renewcommand{\pink}[1]{{%
  \fboxsep=0pt
  \colorbox{red!20}{#1\strut}}}
Oo\pink{p}s!
```

Oops!

I added the `\strut` so that the color would be obvious above and below the highlighted element, rather than covering only the "p". While this isn't really a newbie problem, it's wise to be aware that such possibilities exist, and be ready in such cases to seek expert assistance.

Paragraph endings and vertical mode

The end of a paragraph is a transition from horizontal to vertical mode. A blank line or `\par` will accomplish this transition. It's important to be aware of what mode you're in, since some operations are best performed in vertical mode; the most important is the insertion of floats (figures, tables, algorithms).

Another important consideration is that some features of text are not "frozen" until a paragraph is ended. One important feature is the vertical spacing of baselines, which depends on the font size. Too many newbies try to end a paragraph with a double backslash, resulting in horrors like the following.

```
\Huge Texts with inconsistent descenders
can result in surprises when the font
size changes without a proper paragraph
ending.\\
```

Texts with inconsistent descenders can result in surprises when the font size changes without a proper paragraph ending.

Some environments (but not *all*) are defined with a paragraph break at the end. A problem such as the one shown here won't result in an error or warning message, so adding a proper paragraph break is the proper correction.

The vertical space between paragraphs is determined by the value of `\parskip`; this is set in the document class, but can be reset as needed. But often, it's convenient to add occasional extra space between paragraphs explicitly; this is done with `\vspace` or `\vskip` while in vertical mode (that is, after the blank line or `\par` that ends a paragraph).

The double backslash What a paragraph does *not* end with is the control symbol `\\`. `\\` does end a line. It is the designated command to end lines in tables, poetry, multi-line math environments, and some other situations. But it does not end a paragraph and can trigger a number of error or warning messages.

If `\\` is alone on a line in vertical mode, this error is reported:

```
! LaTeX Error:
    There's no line here to end.
```

Further, if the `\\` is preceded by a (typed) space, in addition to the warning, there may be an extra, unwanted, blank line in the output.

If a line ending with `\\` is very short:

```
Underfull \hbox (badness 10000)
    in paragraph at lines ...
```

This may be okay, but check.

If `\\` is followed by bracketed text, as in `[stuff to be typeset]`, the result will be the mysterious `! Missing number, treated as zero.`
For `\\`, a following (optional) `[...]` is defined to indicate a vertical distance to be skipped; insert `\relax` before the opening bracket.

If extra vertical space *is* wanted after a line broken with `\\`, it can be added by inserting an optional rubber length (usually just a dimension), wrapped in brackets: `\\[`⟨*value*⟩`]`. If such a bracketed expression is really meant to be typeset, it must be preceded by `\relax`.

`\newline` is often a reasonable alternative to break a line.

Font changes

Font changes are a time-honored method of communicating shades of meaning or pointing out distinct or particularly important concepts. Many such instances are built into document classes and packages; for example, theorems are set in an *italic* font, section headings in **bold**, and some journals set figure captions in sans serif to distinguish them from the main text.

LaTeX provides two distinct methods for making font changes. Commands of one class take an argument and limit the persistence of the change to the content of that argument; these have the form of `\textbf{...}` for **bold**, `\textit{...}` for *italic*, etc. The other class sets the font style so that it will not change until another explicit change is made, or it is limited by the scope of an environment; some examples are `{\itshape...}`, `{\bfseries...}`, and `{\sffamily...}`. These commands are best looked up in a good user guide.

Several font-changing commands do different things depending on the context. `\emph{...}` will switch to italic if the current text is upright, or to upright if the current text is italic. Within math, `\text{...}` will set a text string in the same style as the surrounding text; thus, within a theorem, `\text{...}` will be set in italic. If this string should always be upright, like function words, `\textup{...}` should be used instead.

Basic TeX defined two-letter names for most font styles. All of these are of the persistent type. They should be avoided with LaTeX, as some of the LaTeX forms provide improvements, such as a smoother transition between italic and upright type.

Math

Math is always a scoped environment. If started, it must be ended explicitly and unambiguously. Within text, math begins and ends with `$`. LaTeX also provides `\(...\)` for in-text math, but most users stick with the `$`. Many different display environments are defined by the packages amsmath and mathtools, and it is worthwhile to learn them by reading the user guides.

Within math, all input spaces are meaningless to (LA)TeX; they can be entered in the source file as useful to make it readable to a human. Blank lines, however, are considered errors. In both in-text math and displays, the error message will be

```
! Missing $ inserted.
```

This will also result if in-text math is not ended before the paragraph ends, or if a math-only symbol or command is found outside of math mode.

If a blank line occurs in a multi-line display environment from amsmath, the *first* error message will be

```
! Paragraph ended before ⟨env-name⟩
    was complete.
<to be read again>
```

This will be followed by *many* more error messages, all caused by the first. These will be confusing and misleading. Always fix the problem identified by the first error and ignore the rest; they will disappear once the first error is fixed; here, by removing the blank line.

If the appearance of a blank line is wanted for readability, instead use a line with just a `%`.

As with all environments, the `\end` name must exactly match the name specified at `\begin`. A "shorthand" for a single-line, unnumbered display is `\[...\]`. The environments designed for multi-line displays should not be used for a single-line display.

Although LaTeX provided `eqnarray` as a display environment, don't use it. If the display is numbered and the equation is long, the equation can be overprinted by the equation number.

Tables, figures, and other floats

The allowed number of floats, their positions on a page, and the spacing around and between them is defined by the document class. So if something doesn't work as you expect (hope for?), any potential helper will insist on learning what document class is being used.

Input for a float must appear in the source file while there is still enough space on the output page to fit it in. In particular, on two-column pages, a `figure*` or `table*` must occur in the source *before* anything else is set on the page. LaTeX's core float handling does not allow full-width floats to be placed anywhere but at the top of a page; some packages extend this capability, but those won't be discussed here.

Here are the defaults for the basic `article` class.

- Total number of floats allowed on a page with text: 3.
- Number of floats allowed at top of page: 2. Percentage of page allowed for top-of-page floats: 70%.
- Number of floats allowed at bottom of page: 1. Percentage of page allowed for bottom-of-page floats: 30%.
- Minimum height of page required for text: 20%.
- Minimum height of float requiring a page by itself: 50%.

The reference height is `\textheight`. That is, the height of page headers and footers is excluded.

If an insertion is small, must be placed precisely and fits in that location, don't use a float. `\includegraphics` or one of several available table structures should be used directly, often wrapped in `\begin{center} ... \end{center}` (Within a float, use `\centering` instead.)

The wrapfig package supports cut-in inserts at the sides of a page or column. Refer to the documentation for details.

By tradition, captions are applied at the top of tables and the bottom of figures. If an insertion is not a float, the usual `\caption` can't be used. Instead, `\usepackage{caption}` and the command `\captionof`.

The document class and preamble

When embarking on a new document, start by choosing the document class. If the goal is publication in a particular journal, check the publisher's instructions to see what is required. Many popular journal classes are available from CTAN.[2]

If the project is a thesis or dissertation, find out the special requirements, and if your institution provides a tailored class, obtain a copy. Try to determine whether it is actively maintained, and if there is local support. Read the documentation.

It is the responsibility of the document class to define the essential structure of the intended document. If the document you are preparing differs in essential ways from what is supported by the document class, the time to get help is *now*.

There will be features not natively supported by the document class; for example, the choice of how to prepare a bibliography may be left to the author. This is why packages have been created.

Organizing your document Most packages are loaded in the preamble. There is one exception: `\RequirePackage`. This is usually specified before `\documentclass`, and is the place where certain special options should be loaded.

Some authors create a preamble that is suitable for one document, then use the same preamble for their next document, adding more packages as they go. And some unwitting newbies "adopt" such second-hand "templates" without understanding how they were created. *Don't do it!*

Start with a suitable document class and add features (packages, options, and definitions) as they become necessary. Organize the loading of packages into logical groups (all fonts together, for example), and be careful not to load a package more than once; if options are needed, any loaded with a non-first `\usepackage` will be ignored. Some packages automatically load other packages; for example, mathtools loads amsmath and amssymb loads amsfonts. And, very important, pay attention to the order of package loading: hyperref must be loaded (almost) last; the few packages that must come after hyperref are all well documented.

Read the documentation.

Processing the job

Once the file is created, it's time to produce output. There are several engines to choose from: pdfLaTeX, XeLaTeX, and LuaLaTeX. These can be run interactively from the command line, or initiated from an editor. Assuming there are no errors, how many times a document must be processed to produce the final output depends on what features it contains.

In particular, if any cross-references or `\cite`s are present, this information is written out to an `.aux` file; information for a table of contents is written to a `toc` file, and other tables are also possible. The bibliography must be processed by a separate program (and its log checked for errors) with the reformatted bib data written out to yet another file. Then LaTeX must be run (at least) twice more — once to read in the `.aux` and other secondary files and include the bibliography and resolved cross-references, and the second time to resolve the correct page numbers (which will change when the TOC and similar bits are added at the beginning).

All this assumes that there are no errors. Errors will be recorded in the log file. Learn where the log file is located, and make a habit of referring to it. Warnings, such as those for missing characters, will also be recorded there, but may not be shown online:

```
Missing character: There is no ⟨char⟩
                           in font ⟨font⟩!
```

In the log, some errors may appear with closely grouped line numbers. If so, and the first is one that interrupts the orderly processing of a scoped environment, following errors may be spurious. So fix the first error and try processing before trying to understand the others; often, they may just go away.

Good luck. With practice comes understanding.

Oh. . . Remember to read the documentation.

Acknowledgment

Thanks to samcarter, Mikael Sundquist, and (as always) Karl Berry for suggestions and for finding and exterminating my typos. I can find them in other people's writing, but often not in my own.

References

[1] Barbara Beeton. Debugging LaTeX files — Illegitimi non carborundum, *TUGboat* 38:2, 159–164 (2017). tug.org/TUGboat/tb38-2/tb119beet.pdf

◇ Barbara Beeton
TUGboat
Providence, RI, USA
bnb (at) tug dot org

[2] ctan.org/search

Architectural guides for Bonn — book production with ConTEXt

Henning Hraban Ramm

“Architekturführer der Werkstatt Baukultur Bonn”

At the University of Bonn, there is a group of scholars who care about the modernist buildings that were built after the second world war, when Bonn was Germany’s capital. They do research, offer guided tours and also publish a series of little architectural guides. My publisher colleagues were already involved with them when they studied German literature in Bonn, so we took over this series when we founded Dreiviertelhaus publishers in 2017. As it happened, I took over the design even earlier, since their designer had no time after becoming a mother.

The design is rather simple, so I decided to do it in ConTEXt (instead of InDesign). But the structure of every booklet is unique, since they have a wide variety in the contents: Some volumes are about only one building, others about an ensemble or a housing estate or a themed collection such as “buildings of the university”; some are by a single author, others collect contributions by several authors. That means I must adapt the table of contents and the structure of titles in every other volume. (Figure 1 is an example.)

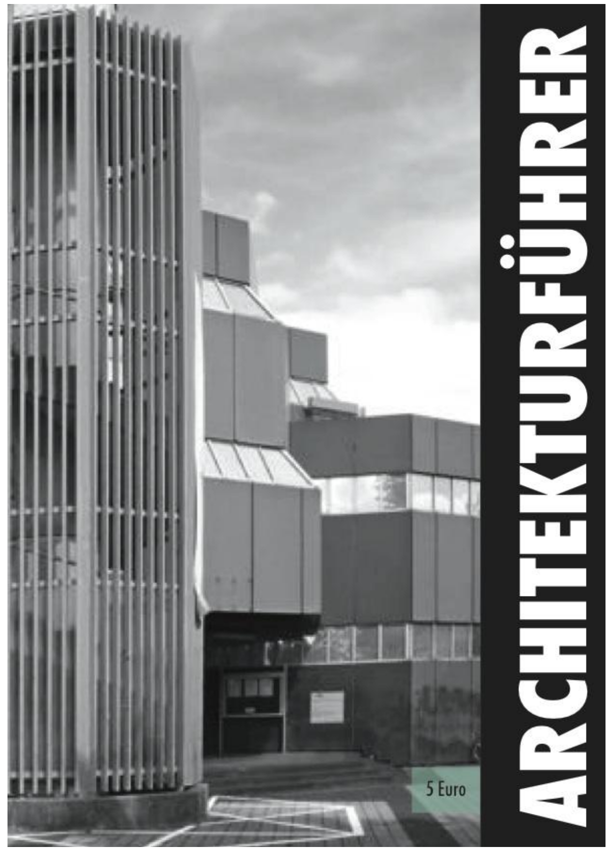


Figure 1: symbolic #0 guide

Design and layout

Since we use many photos from archives, most pages are black-and-white, as well as the front covers. But the booklets are printed in color, because we also show current pictures, and sometimes color is important, for example with stained glass windows in churches or other artwork.

For the cover, we try to use one image front-to-back, but it’s just not possible for every volume to find a landscape photo where the important part of a building is on the right-hand side. Since my setup expects one image, I glue different photos for front and back together in such cases. (Figure 2.)

Figure 2: cover images (back and front)

Interior images are often full page, sometimes even over a double page spread.

Images that cover the full width or height of a page need to be a few millimeters bigger to avoid problems in paper trimming; this is called *bleed*. This affects not only full page images, but everything that touches paper edges. We also have images that stay within the type area.

Most images have captions. On full page images, the caption is moved into the image and gets a background shadow to increase readability. (Figure 3.)

Henning Hraban Ramm
doi.org/10.47397/tb/44-2/tb137ramm-books

Die Baugeschichte

Am 19. April 1951 erfolgte die Grundsteinlegung für die Siedlung in Tannenbusch, einige Wochen später dann auch in Muffendorf. Die Grundstücksverhandlungen mit der Erbengemeinschaft von Carstanjen für die Siedlung in der Plittersdorfer Aue hatten am 2. Januar 1951 begonnen. Aufgrund des Termindrucks kamen die Amerikaner bei der Preisgestaltung den Grundeigentümern entgegen. So konnte schon am 15. Februar 1951 der erste Spatenstich für die HICOG-Siedlung in Plittersdorf erfolgen. Bereits an diesem Tag waren Hunderte von Bauarbeitern und Handwerkern auf der Baustelle in Plittersdorf, wenig später waren etwa 3200 Arbeiter dort beschäftigt. Es war zu diesem Zeitpunkt die größte Baustelle in Westdeutschland. Am 1. Juni 1951 fand in Plittersdorf das Richtfest für alle drei Siedlungsbaustellen und für das Verwaltungsgebäude der HICOG, das in der Deichmanns Aue in Mehlem gebaut wurde, statt.

Alle Bauarbeiten standen unter hohem Zeitdruck. Der ursprünglich geplante Fertigstellungstermin 1. September 1951 konnte wegen des verspäteten Baubeginns der beiden Siedlungen für die deutschen Mitarbeiter nicht gehalten werden. Der neu anvisierte Termin war der 1. November 1951. Zu diesem Zeitpunkt waren die Siedlungen weitgehend bezugsfertig, ebenso die fünf Villen an der Turmstraße für obere HICOG-Beamte im Generalsrang. In der Plittersdorfer Siedlung waren zusätzlich zu den Wohnungen auch die Einrichtungen der Infrastruktur – Hallenbad, Bowling-Bahn, Kino, Tank-

14

Bauarbeiten in Plittersdorf

stelle, Heizwerk, Supermarkt – fertig. Tatsächlich begann am 5. November 1951 der Umzug der HICOG-Beschäftigten nach Bonn. Etwa 560 US-Angestellte mit rund 430 Familienangehörigen zogen nach Plittersdorf, etwa 620 deutsche HICOG-Angestellte mit rund 920 Familien-Angehörigen nach Tannenbusch beziehungsweise Muffendorf.

Der Umzug war generalstabsmäßig geplant. Jeder HICOG-Mitarbeiter wurde zehn Tage vor dem Umzugstag über den Termin informiert, erhielt fünf Tage vor dem Termin den Schlüssel zu der Bonner Wohnung, und zwei Tage vor dem Umzugstag kam eine Packer-Kolonne. Die Möbelwagen fuhren nach der Beladung in Frankfurt über die Autobahn nach Bonn und

15

Ausstellungshalle (2007)

Literatur

Abresch, Bettina: *Exkurs: Der Kleine Königshof*, in: Margarethe Jochimsen (Hg.): Bonner Kunstverein 1963–1993, Bonn 1993, S. 202 f

Borger, Hugo: *Bonner Kunstverein 1967–1973*, in: Margarethe Jochimsen (Hg.): Bonner Kunstverein 1963–1993, Bonn 1993, S. 19

Conradi, Peter (Hg.): *Kultur und Gesellschaft. Vorträge und Diskussionsbeiträge aus dem Gesprächskreis Kultur und Gesellschaft der Friedrich-Ebert-Stiftung. Architektur als soziale Verpflichtung kontra Architektur als Kunst – Bauhausgedanke und Postmoderne, ein unversöhnlicher Gedanke?* (Architektur und Stadtplanung, 2), Bonn 1987

Denk, Andreas: *Mehr Kunst für Bonn – Eine Bürgerinitiative für die Kunst*, in: Margarethe Jochimsen (Hg.): Bonner Kunstverein 1963–1993, Bonn 1993, S. 142–150

Denk, Andreas / Flagge, Ingeborg: *Kunsthalle*, in: Andreas Denk / Ingeborg Flagge (Hg.): Architekturführer Bonn, Berlin 1997, S. 65

Hackelsberger, Christoph: *Vortrag vom 9. Juli 1987*, in: Peter Conradi (Hg.): Kultur und Gesellschaft. Vorträge und Diskussionsbeiträge aus dem Gesprächskreis Kultur und Gesellschaft der Friedrich-Ebert-Stiftung. Architektur als soziale Verpflichtung kontra Architektur als Kunst – Bauhausgedanke und Postmoderne, ein unversöhnlicher Gedanke? (Architektur und Stadtplanung, 2), Bonn 1987, S. 5-14

51

Figure 3: example spreads with half-page and full-page images

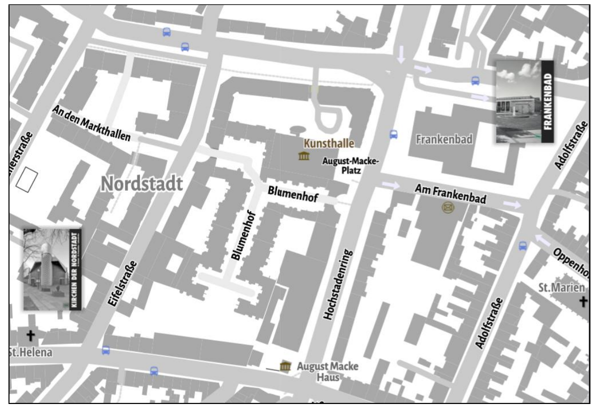


Figure 4: city map, with advertising for related guides

Maps

There's another tricky subject in these architectural guides, namely city maps.

For my architectural guide on the Kyrgyz capital Bishkek, I got experience with processing OpenStreetMap data for custom maps.

I'm using Maperitive,[1] because it allows for batch processing. Maperitive is written in .NET, and I run it on my Mac with Mono. It's horribly slow, the programming interface is severely underdocumented, and the latest version is from 2018, but it's still the best choice and I somehow manage to get what I want.

What I want is also a custom style with very subdued colors, nearly black and white, and not many location markers for shops etc. Maperitive uses style sheets that are somewhat similar to CSS, so you have selectors and style instructions.

The output is SVG, and I use Inkscape to convert it to PDF for inclusion. ConTEXt LMTX doesn't need this any more and can process SVG on its own via a MetaPost conversion. But when I made the latest architectural guide in 2019, this was not yet possible. Also, I want to postprocess the images, e.g. deleting unnecessary labels.

You may have read in *TUGboat* 42:3 that ConTEXt can process OpenStreetMap data on its own, also via a MetaPost conversion. This is true, but unfortunately not more than a proof of concept. It can't handle labels, like street names, so it's quite useless for a city map. The colors are ugly, too — that would be easy to change, and I promised to provide a theme, but the rendering is just not flexible enough: All paths can only be drawn as single lines, while if you look at other OpenStreetMap renderers, streets usually have a fill and an outline, and for railway tracks you need a thick white line with a dashed black line on it. My programming skills don't suffice to fix that.

So I stick to my proven workflow for the time being. (Figure 4.)

Setup

I wrote the setup for these architectural guides mostly in 2015, and since then, plenty has happened — not only has ConTEXt moved to LuaMetaTEX, but also I've learned a lot and can do a bit better, so I found my old code a bit embarrassing and refactored it, just in time for the upcoming guides that we hope to publish in 2023/24.

I will leave out all the setups with regard to language, fonts and colors.

Simple page layout First we define the page size. That's easy:

```
\setuppapersize[A6]
```

The page layout is quite simple, we have no page header and usually don't need footnotes.

If you set up a layout in ConTEXt, you should always define the parameters `backspace` and `width` first, then `topspace` and `height`. The latter includes

[1] `https://maperitive.net`

header and footer. You can leave the other areas like margins and edges alone if you don't need them.

Header and footer setting reflect that we don't need page headers and the footer only for page numbers. We need double pages to get the page numbers in the outer footer, otherwise we couldn't distinguish left and right pages.

```
\setuplayout[
  backspace=12.5mm, width=80mm,
  topspace=12.5mm, height=125mm, % text+footer
  header=0mm,footer=10mm,
]
\setuppagenumbering[
  alternative=doublesided,
]
```

Bleed and trim Most of our images cover the full page width, and that means they must *bleed*. 3 mm is a traditional value; in this small format, 1 mm probably would be enough. If our printshop tells me to change it, I want to change it in only one place.

```
\definemeasure[Bleed][3mm]
\definemeasure[Trim][7.5mm]
\setuplayout[
  marking=on, % cut marks
  location=middle,
  bleedoffset=\measure{Bleed},
  trimoffset=-\measure{Trim},
]
```

With regard to printing, we activate cut marks and center the page on the sheet. The trim offset is the difference between sheet and page size as a negative value. The bleed offset is from the page outward as a positive value. It's the same on all sides.

If you would check the outcome so far, you couldn't find these boxes in the PDF. The activation is strangely coupled to some PDF viewer settings:

```
\setupinteractionscreen[
  option={doublesided,bookmark},
    % necessary for Trim/BleedBox:
  width=max,height=max,
]
```

This should work now. But what's the sheet size? We only defined the paper size! Let's fix this:

```
\setuppapersize[A6][A6,oversized]
```

The `oversized` option adds 7.5 mm around the A6 page. We could also define that size explicitly or use the envelope size C6 instead. (Figure 5.)

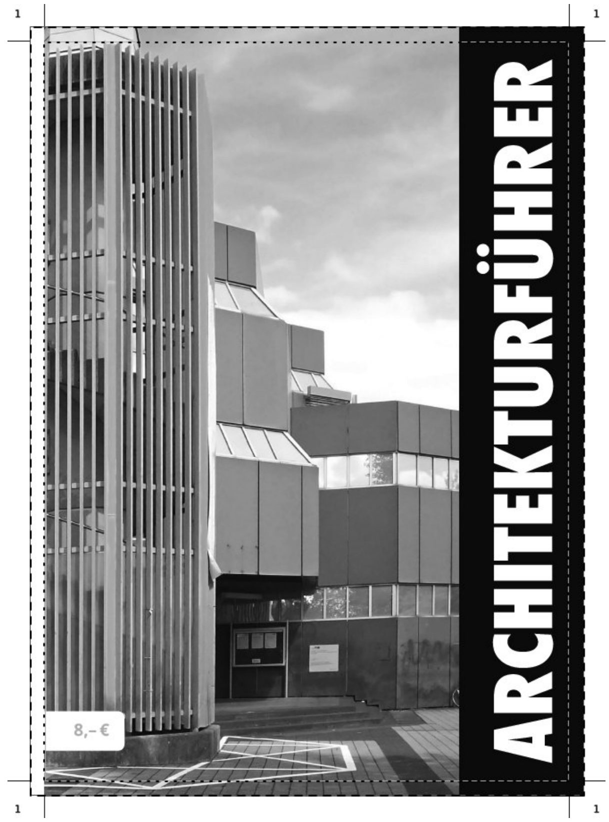


Figure 5: title page with crop marks, trim box (inner) and bleed box (outer)

Preview and print mode While we need bleed, trim and cut marks in the PDF for the printshop, they might confuse the authors in the preview version. They're also not needed for an ebook.

So let's mode-ify the settings. It turns out we only need one mode, 'print'; if activated, it fixes the page size, bleed and trim; it can also turn off interaction (links, etc.).

Another topic where it makes sense to distinguish between preview and print mode is image resolution. It makes no sense to send draft PDFs with high resolution images, and some pictures could use some downsampling even in print mode.

```
% preview (correction copies)
\startnotmode[print]
  \setuppapersize[A6]
  \def\Resolution{96}
  \setupinteraction[state=start]
  \setupexternalfigures[
    conversion=lowres.jpg,
  ]
  % no bleed/trim settings
\stopnotmode

% print version
```

```
\startmode[print]
  \setuppapersize[A6][A6,oversized]
  \def\Resolution{200}
  \setupinteraction[state=stop]
  \setupexternalfigures[
    conversion=hires.jpg,
  ]
  % setuplayout with bleed/trim as above
\stopmode

\setupexternalfigures[
  directory={img},
  resolution={\Resolution},
]
\loadluafile[grph-downsample]
```

This resolution stuff is not (or not yet) a feature of ConTEXt, but handled by some Lua functions that call GraphicsMagick during the TEX run to reduce the image size.

Color conversion to greyscale *is* already included in ConTEXt and works the same way, but here we don't need a greyscale mode.

Image dimensions For our image calculations besides resolution, we need a few basic dimensions.

```
\definemeasure[maxWidth]
  [\paperwidth + \measured{Bleed}]
\definemeasure[maxHeight]
  [\paperheight + 2\measured{Bleed}]
\definemeasure[doubleWidth]
  [2\measured{maxWidth}]
% offsets of images from the type area
\definemeasure[topOffset]
  [\topspace+\headerheight+\measured{Bleed}]
\definemeasure[bottomOffset]
  [\bottomheight+\footerheight+\measured{Bleed}]
```

Where you would use `\newdimen` and `\dimexpr` in ε-TEX, you should use `\definemeasure` in ConTEXt. My companion article "Calculating covers" in this issue (pp. 176–179) explains dimension calculations.

Layers for image placement If you want to place elements in specific locations, the ConTEXt way is to use layers.

For images, it makes sense to use full page layers, but we need to distinguish right and left pages.

```
\definelayer[bgpicleft][
  x=-\measure{Bleed},y=-\measure{Bleed},
  width=\measure{maxWidth},
  height=\measure{maxHeight},
] % incl. bleed
\definelayer[bgpicright][
  x=0mm,y=-\measure{Bleed},
  width=\measure{maxWidth},
  height=\measure{maxHeight},
] % incl. bleed
\setupbackgrounds[leftpage]
  [background=bgpicleft]
\setupbackgrounds[rightpage]
  [background=bgpicright]
```

After definition, we must assign the layers as backgrounds. It's possible to use several layers for one area: `background` takes a list, left to right is top to bottom.

Cover layers For the cover, we need additional layers, and we can already set up the black bar as a text background. (Figure 6.)

```
\definelayer[titlebar][
  x=83mm,y=-\measure{Bleed},
  width=25mm,
  height=\measure{maxHeight},
]
\setupframed[frame=off,offset=overlay]
\setlayerframed[titlebar][
  background=color,
  backgroundcolor=titlebarcolor,
  width=25mm,
  height=\measure{maxHeight},
]{\strut}
```

Image placement

Sorry, I won't show you the implementation of my macros — it's long, convoluted, and ugly.

Full page images The placement command for a full page image looks like this:

Figure 6: #13 HICOG settlements

```
\startpostponing[15]
\pagefig
  [fig:10544-08]% reference
  [rh]% placement code
  {Kurpark, 1950er Jahre}% caption
  {DA01_10544-08}% image file
\stoppostponing
```

"Postponing" moves content to a specific page, the page number can be absolute or relative (+1). Due to expansion and buffering issues it's not possible to include this in a macro.

The \pagefig macro is my own; it takes a reference, a placement code, a caption and the filename of an image. But what does it do?

- decide if we're on a right or left page
- start an empty "makeup" (special layout page)
- place the picture on the layer for the left/right page
- clip the picture to fit (placement code defines if height or width are leading)
- place the caption in the footer (usually white on a dark shadow)
- place debugging information (e.g. filename) in the trim area

The code for a double page image looks nearly the same:

```
\startpostponing[+0]
\doublepagefig
  [fig:11390-29]
  [lh]
  {Blick von Osten}
  {DA01_11390-29}
\stoppostponing
```

This instance was placed between chapters and uses "immediate" postponing (+0).

The macro works similarly to the previous one, except we place the left half of the picture on the layer for the left page and the right half on the right page, each in its own makeup. (A multi-page makeup would confuse the page numbering.) The placement code defines the location of the caption.

Half-page images The call for an image that does not cover a whole page looks like this:

```
\topfig
  [fig:9251]
  [rw]
  {Großer Saal}
  {IMG_9251}
```

I love a consistent interface. But the macro works differently:

- decide if we're on a right or left page
- calculate the actual image dimensions with a Lua function
- decide where to clip (top/bottom) according to placement code
- calculate how much to clip so that the picture fits the line grid
- place it as a float, but move it into the trim area

Why the calculations? I'm using grid setting, even if this is rather questionable with these picture-heavy booklets. But it implies that all images should "sit" on a grid line, i.e. a baseline of body text. ConTEXt couldn't do that on its own. (Only recently, Hans Hagen extended the options for float placement; it might be easier now.) Also, the top border of an image should align with the x-height of a text line, but that doesn't matter in this case.

The image has a fixed width, namely the page width plus bleed. With proportional scaling, we know its maximum height. We subtract the space above the type area (4 values) plus bleed. The remainder modulo the line height is what we need to cut.

It would have also been possible to just move the image, without clipping it.

The simplified float placement looks like this:

```
\startplacefigure[
 location={top,high},
 reference={#1},
 title={#3},
]
\offset[
 topoffset=-\topOffset,
 leftoffset=\measure{leftOffset},
]{%
  \clip[
   x=0mm,y=\topCut,
   width=\measure{maxWidth},
   height=\measure{calculatedImgHeight},
  ]{%
   \externalfigure[#4][width=\measure{maxWidth}]
  }%
}%
\stopplacefigure
```

Shadow captions

The shadow behind captions in full-page images is a MetaPost graphic: A number of stacked rounded rectangles of slightly increasing size, set to a high transparency in "multiply" mode, so that the main shape becomes dark and the borders get blurry.

This graphic is set as an overlay and used as a background to the (invisible) caption frame.

```
\startuniqueMPgraphic{mpshadow}
mw := BodyFontSize/3;
ox := -0.5 ;  % offset x
oy := -0.5 ;  % offset y
bx := 1.5mw ; % bleed x (height of the shadow)
by := 1.5mw ; % bleed y (width of the shadow)
```

Figure 7: multiline caption with a subtle shadow

```
rx := 3mw ;   % max. corner radius x
ry := 2mw ;   % max. corner radius y
steps := 10 ; % number of shadow layers,
              % 10 is a good value
hue := 0.015 ; % 0.02 is a good value
ycorr := 1mw ; % difference between overlay
               % height and shadow height
for step = 1 upto steps:
  part := (step-1)/steps;
  xstep := bx * part ; % current part of bleed
  ystep := by * part ;
  crx := (rx + rx*part)/2; % current radius
  cry := (ry + ry*part)/2;
  % points of the rounded rectangle
  xa := -xstep + ox;
  xb := -xstep + ox + crx;
  xc :=  xstep + ox - crx + \overlaywidth;
  xd :=  xstep + ox       + \overlaywidth;
  ya := -ystep + ycorr + oy;
  yb := -ystep + ycorr + oy + cry;
  yc :=  ystep - ycorr + oy - cry
                            + \overlayheight;
  yd :=  ystep - ycorr + oy + \overlayheight;

  fill (xb, ya)---(xc, ya)...(xd, yb)---
    (xd, yc)...(xc, yd)---(xb, yd)...
    (xa, yc)---(xa, yb)...cycle
    withcolor transparent(1, hue, black) ;
endfor;
setbounds currentpicture to OverlayBox ;
\stopuniqueMPgraphic
\defineoverlay[shadow][\useMPgraphic{mpshadow}]
% ...
\inframed[frame=off,
  background=shadow,
  foregroundcolor=white,
]{This is my caption.}
```

This was first made for a shadow behind images, and it works well for text when there's only one line or if you can make all lines the same width. The example is one of the few where that wasn't possible, but I was never satisfied with this solution.

The outline approach Just recently I found out how to make a shadow that adapts to the font shape. This uses a LuaMetaFun extension for font outlines. Again, we stack elements with a low opacity, this time with an increasing outline "rulethickness".

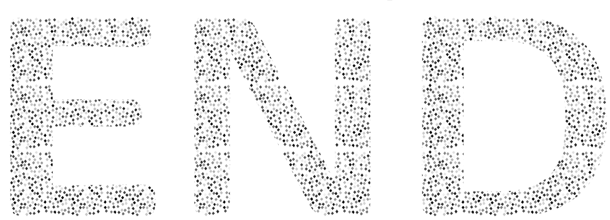
Now, doesn't this look better?
The code is much shorter,
and it also works with several lines.

The shadow color is somewhat irregular due to overlapping outlines between letters or letter elements. Maybe it's possible to combine the paths.

```
\definecolor[tshade][t=.05,a=1,k=1]
\starttexdefinition ShadowText #1
\startMPcode
steps := 10 ; % number of shadow layers
rulesize := BodyFontSize/steps/3;
for step = 1 upto steps:
  draw lmt_outline [
    text = "\vbox{\strut #1}",
    kind = "fillup",
    fillcolor = "tshade",
    rulethickness = (step*rulesize),
  ];
endfor;
% finally, opaque white text on top
draw lmt_outline [
  text = "\vbox{\strut #1}",
  kind = "fillup",
  fillcolor = "white",
  rulethickness = 0,
];
\stopMPcode
\stoptexdefinition
% ...
\ShadowText{Now, doesn't this ...}
```

If you use this with big text, it makes sense to add `randomized 3` to the `lmt_outline` call to make it look a bit more natural.

The LuaMetaFun `lmt` functions were introduced in 2021 and are quite fun to play with. E.g. to fill an `lmt_outline` path with an `lmt_poisson` pattern:



◇ Henning Hraban Ramm
Limburg, Germany
hraban (at) fiee dot net

Calculating covers with ConTEXt

Henning Hraban Ramm

Abstract

Every TEX user can typeset a book, but the cover might be a different story. We will learn a bit about dimensions and calculations as we calculate a cover.

Take cover!

I used to create the covers for my ConTEXt books in a graphics application. I still think that's the best way to plan a cover, because I can try new ideas or make changes, and see the results straight away.

But there are disadvantages. It can take a lot of steps to make a small change. You need to hand calculate the spine width, and remember to change it if the number of pages changes. If you change the title or author on the front cover, you need to remember to change it on the spine too. It can get messy if you want several graphics, each slightly different.

I want to show you how I use ConTEXt to calculate covers, without those disadvantages.

Basic setup

Let's start simple: we define a paper size and a whole-page layer, on which we will place our elements:

```
\definepapersize[Cover][width=350mm,height=240mm]
\setuppapersize[Cover]

\definelayer[cover][
  x=0mm,y=0mm,
  width=\paperwidth,
  height=\paperheight,
]
\setupbackgrounds[page]
    [background=cover,state=start]

\starttext
\setlayer[cover][
  x=200mm,
  y=20mm,
]{\ss\bfd My Title}
\strut
\stoptext
```

The `\strut` at the end is necessary, otherwise the page has no content and ConTEXt won't even display the content of the layer.

Dimensions

Where did we get those dimensions? I used the size of the DANTE books. Their page is 165 by 240 mm, i. e. their cover width is two times 165 mm plus a 20 mm spine. Maybe our printshop told us these dimensions, but we would need to ask again if something changes. Let's see if TEX can do the calculations.

```
\definepapersize[Cover][
  width={2 * 165mm + 20mm},
  height=240mm,]
```

That would have been nice, but ConTEXt complained about an "Illegal unit of measure". It's not as easy as it might be.

For ConTEXt to calculate with dimensions, we must use 'dimension expressions'. As a rule, they must start with a dimension:

```
\definepapersize[Cover][
  width=\dimexpr 165mm * 2 + 20mm\relax,
  height=240mm,]
```

Do we need to use such cumbersome code for every element on the cover? No! We can predefine the most important values, most simply as macros:

```
\def\PageWidth{165mm}
\def\SpineWidth{20mm}

\definepapersize[Cover][
  width=\dimexpr \PageWidth*2 + \SpineWidth\relax,
  height=240mm,]
```

That works, but it's cleaner and more reliable if we define our own dimensions. In ε-TEX it looked like this:

```
\newdimen\PageWidth
\PageWidth=165mm
```

But because we use ConTEXt, it should look like ConTEXt:

```
\definemeasure[PageWidth][165mm]
```

You can retrieve such a value in two ways: with `\measure` as a string for assignments in `\setup` commands, or with `\measured` as a dimension for calculations.

In the second argument of `\definemeasure`, we can execute calculations without writing an explicit `\dimexpr`, but internally, since `\definemeasure` uses `\dimexpr`, it has the same limitations:

```
\definemeasure[CoverWidth]
    [2\measured{PageWidth} + 20mm]
```

Oh, this expression doesn't start with a dimension!? Well, simple factors like this are possible, sometimes even with decimal numbers. For example, these are valid: '`2\lineheight`', '`1.5\lineheight`', and '`\lineheight * 2`', but '`\lineheight * 1.5`' is not. In such cases you can cheat with fractions: '`*1.5`' throws an error, while '`*3/2`' works.

Now the complete code is:

```
\definemeasure[PageWidth][165mm]
\definemeasure[PageHeight][240mm]
```

First published in German in DANTE's *Die TEXnische Komödie* 1/2023, pp. 16–25; translation by the author. English version originally edited by Peter Hopcroft.

doi.org/10.47397/tb/44-2/tb137ramm-covers

```
\definemeasure[SpineWidth][20mm]
\definemeasure[CoverWidth]
  [2\measured{PageWidth} + \measured{SpineWidth}]

\definepapersize[Cover][
  width=\measure{CoverWidth},
  height=\measure{PageHeight}]
\setuppapersize[Cover]

\definelayer[Cover][
  x=0mm,y=0mm,
  width=\paperwidth,
  height=\paperheight,]
\setupbackgrounds[page]
    [background=cover,state=start]

\starttext
\setlayer[Cover][
  x=\dimexpr\measured{PageWidth}
           + \measured{SpineWidth}
           + 15mm\relax,
  y=20mm,
]{\ss\bfd My Title}
\strut
\stoptext
```

Page count

If the number of pages changes, we only need to change one number. But it is better to calculate it automatically:

```
\useexternalfigure[content][book.pdf]
\getfiguredimensions[content]
\expanded{\definemeasure[SpineWidth]
    [2mm + (0.09mm * 3/2 * \noffigurepages/2)]}
```

`\useexternalfigure` gives our content PDF the symbolic name `content`. `\getfiguredimensions` detects the properties of the current image, including the number of pages in the PDF, which is stored as `\noffigurepages`. We have to use `\expanded` to execute `\definemeasure` immediately. Otherwise the current image would have changed, and `\noffigurepages` would be wrong.

Paper thickness

How did we get that formula? We need the number of sheets. And my vocational teachers used to remind us: "Paper has two sides!" Our book will be printed on 90 gsm[1] paper. Standard 90 gsm paper is 0.09 mm thick. Our paper has some light filler material, and is thicker: 1.5 times 0.09 mm thick. The 1.5 (written above as 3/2) is called bulk. Of course, we could define paper thickness as a ConTEXt dimension, but we don't need it elsewhere.

The 2 mm above is a fold allowance, about 1 mm for each fold between a cover and the spine. Strictly speaking, the fold allowance calculation should take into account the thickness of the cover cardboard.

Now our cover automatically adapts to the number of pages in the book. Not too bad.

[1] grams per square meter, also known as 'grammage'

Layers

We want to place the title on the front cover, as well as the subtitle and author. We want these on the spine as well. On the back cover, we want some blurb and an ISBN barcode. Must we repeat the same laborious calculations time and again? No!

We will define separate layers for front cover, back cover, and spine. Then we can give the offsets of elements relative to their parent layer:

```
% ...
\definelayer[BC][ % back cover
  hoffset=0mm,
  y=0mm,
  width=\measure{PageWidth},
  height=\measure{PageHeight},
]
\definemeasure[FrontStart]
  [\measured{PageWidth} + \measured{SpineWidth}]
\definelayer[FC][ % front cover
  hoffset=\measure{FrontStart},
  y=0mm,
  width=\measure{PageWidth},
  height=\measure{PageHeight},
]
\definelayer[Spine][
  hoffset=\measure{PageWidth},
  y=0mm,
  width=\measure{SpineWidth},
  height=\measure{PageHeight},
]
\setupbackgrounds[page]
    [background={Cover,BC,FC,Spine},
     state=start]
% ...
\setlayer[FC][
  x=15mm,
  y=20mm,
]{\ss\bfd My Title}
\setlayerframed[Spine][
  y=12mm,
  offset=overlay,
  frame=off,
  align=flushleft,
  width=\measure{SpineWidth},
  height=0.66\measured{PageHeight},
]{%
  \rotate[
    rotation=90,
    height=\measure{SpineWidth},
    width=0.66\measured{PageHeight},
    align={lohi,flushright},
  ]{Author: Title}%
}
```

For the spine text I used `\setlayerframed` so we have all the options of `\framed` to hand. While planning a cover, I like to turn on the frames to check the positions of the elements. We can make this setting a command-line argument:

```
\setupframed[offset=overlay] % no border distance
\startnotmode[debug]
  \setupframed[frame=off]
\stopnotmode
```

To turn the frame on, call ConTEXt with the option `--mode=debug`. The above code also sets `offset=overlay` for every frame, so we don't have to do this for every frame individually.

Buffers

Next, let's take care of the blurb. We can place it with `\setlayerframed[BC]`. But I find it confusing to have long text strings loitering within ConTEXt calculations. Therefore, we define the text as a buffer in advance:

```
\startbuffer[Blurb]
\quotation{I never read a better book!}
\wordright{(M. Reich-Radecki)}
\blank[2*line]
Something about the brilliant content...
\stopbuffer

\startsetups[blurb]
% font/alignment/indent settings
\stopsetups

% ...
\setlayerframed[BC][
  x=.15\measured{PageWidth},
  y=20mm,
  width=.7\measured{PageWidth},
  height=.8\measured{PageHeight},
  setups=blurb,
]{\getbuffer[Blurb]}
```

Variables

Let's define some book data as variables, all in one place:

```
\setvariables[book][
  contentPdf={vol01},% name of the content file
  author={Donald E. Knuth},
  title={The \TeX book},
  subtitle={about command-based typesetting},
  series={Computers & Typesetting},
  volume={A},
  isbn={978-3-12345-007-Z},
  coverimage={lion},
]
% ...
\useexternalfigure[content]
    [\getvariable{book}{contentPdf}]
\getfiguredimensions[content]
% ...
\setlayer[FC][
  x=15mm,y=20mm,
  setups=maintitle
]{\getvariable{book}{title}}
```

When we change any of the above book data, it automatically changes on all the layers where it appears. Wonderful. I have seen books with a different title or author on the front cover and on the spine.

As you may observe, we can use expressions like `\getvariable{book}{title}` to retrieve values. Of course, we also could have used macros.

Environment

Since we need the data for the book's content (e. g. fly title, imprint) as well, we should save it to an external environment file that we can load in both the cover and the content documents:

```
\startenvironment settings
\project bookbook

\setvariables[book][
%...
]
\stopenvironment
```

Bleed

Next step: we'll add a background image. Since it should cover the whole page, we must set it up to 'bleed'. That means that the image extends a few millimetres past where the book will be trimmed. Otherwise there can be white gaps at the edges if the printed book isn't cut exactly to the trim line. (That's usually because paper changes size as humidity changes during printing, rather than the fault of the printshop or bookbinder.)

```
\definemeasure[Bleed][3mm]
\definemeasure[MaxHeight]
    [\measured{PageHeight} + 2\measured{Bleed}]
\setlayerframed[FC][
  %x=-\measured{Bleed},
  y=-\measured{Bleed},
]{\externalfigure[\getvariable{book}{coverimage}]
                 [height=\measure{MaxHeight}]}
```

Of course, we can also put a background image for the complete cover (back, spine, and front) on the "Cover" layer. If the number of pages changes, the width of the image will change slightly. Usually this doesn't matter.

ConTEXt documents often use MetaPost graphics as background images. For those, you can use the variables `OverlayWidth` and `OverlayHeight`.

While the image now has bleed, we can't see it when we look at the PDF on-screen, because we see the trimmed paper size. We can use the `oversized` option to expand the paper size by 7.5 mm on all sides:

```
\setuppapersize[Cover][Cover,oversized]
```

But because we also need the dimension for our calculations, we will expand the paper size explicitly:

```
\definemeasure[Trim][7.5mm]
\definemeasure[CoverWidth]
  [2\measured{PageWidth} + \measured{SpineWidth}]
\definemeasure[CoverWidthPlus]
  [2\measured{PageWidth} + 2\measured{Trim}
   + \measured{SpineWidth}]
\definemeasure[CoverHeightPlus]
  [\measured{PageHeight} + 2\measured{Trim}]

\definepapersize[Cover][
  width=\measure{CoverWidth},
  height=\measure{PageHeight}]
\definepapersize[CoverPlus][
  width=\measure{CoverWidthPlus},
  height=\measure{CoverHeightPlus}]
\setuppapersize[Cover][CoverPlus]
```

We don't need to change the layers — their elements don't get trimmed at their borders.

Now we also want to see crop marks, and while we're at it, we should properly set up the invisible 'boxes' in the PDF that outline the trimmed area (TrimBox) and bleed area (BleedBox). You can only *see* them in Acrobat (Preferences > Page Display > Show art, trim & bleed boxes), otherwise you can check the values with `pdfinfo -box`.

```
\setuplayout[
  marking=on,% crop marks
  location=middle,% center page on the sheet
  cropoffset=0mm,
  bleedoffset=\measure{Bleed},
  trimoffset=-\measure{Trim},
]
\setupinteractionscreen[width=max,height=max]
```

- A positive value of `cropoffset` shrinks the visible area and also affects both of the other values.
- A negative value of `trimoffset` defines the offset from TrimBox to CropBox.
- A positive value of `bleedoffset` defines the bleed as the offset from BleedBox to TrimBox.
- Only `\setupinteractionscreen` activates the settings.

Setting TrimBox and BleedBox in this way does not affect the positions of the layers or their contents.

More hints about dimension calculations

Dimension expressions (`\dimexpr`) can be nested. It sometimes makes sense to call `\dimexpr...\relax` within a `\dimexpr`.

Internally, TeX calculates with integer 'scaled points' (`sp`) of 1/65536 pt. The maximum value for dimensions is 16384 pt (about 5.75 m).

If we output dimension values using `\measure`, they get typeset in pt (TeX points). We can convert units like this:

```
\define[2]\Conv{\scratchdimen #1 \the\nodimen #2
                \scratchdimen}
% first parameter: dimension,
% second parameter: unit. For example:
\Conv{1pt}{mm}
```

Final remarks

The code that I use in my publishing house also handles optional flaps.

This article is about softcovers. For hardcovers you need a bigger cover and more bleed (about 15 mm), because the cover paper gets glued around the cover cardboard. The spine also needs more folding allowance (about 4–5 mm) for the hinges. You can change the calculations above.

◇ Henning Hraban Ramm
hraban (at) fiee dot net

New dimensions: Edith and Tove

Willi Egger, Hans Hagen, Edith Sundqvist, Mikael P. Sundqvist

Male dominance

When you start using TeX you can't get around the fact that it uses dimensions. You have to set up a paper size, configure a line width, tell it what font size to use, etc. As with many techniques that evolved in different countries the way to express a dimension can be done differently. In Europe we like to talk in centimeters (`cm`) or millimeters (`mm`) and in the United States it's all about inches (`in`). Typographers all over the world speak in terms of points (`pt`), didots (`dd`), ciceros (`cc`) and picas (`pc`) while those messing around with digital typography prefer "big" (PostScript) points (`bp`). TeXies sometimes like scaled points (`sp`) as 1 `sp` is the smallest internal representation of a unit. When someone talks "points" you can't be sure if it is big points or TeX points because the `pt` unit is often used for both.

There are also font-related units, like the popular em width (`em`) and ex height (`ex`) and there is even a pixel unit (`px`) that can be set to some resolution but that one is rarely used. There is also a math unit (`mu`) that scales with the math font in use.

All units are internally scaled points and one real point is 65536 scaled points. That means that when a unit is entered it gets mapped onto this internal scaled point quantity.[1]

For a while we had the new didot and new cicero but in LuaMetaTeX these were dropped because no one used them. On the contrary, the recently introduced (Don) Knuth unit (`dk`) is quite convenient and we use it as a convenient offset for a so-called TeX page environment, which we use a lot in testing math functionality (Hans Hagen, A new unit for LMTX: The `dk`; `tug.org/TUGboat/tb42-3/tb132hagen-dk.pdf`).

From this summary we can observe that there are three units that are names: Didot, Cicero and Knuth. But do you realize that these are all males? That can't be right and should be fixed. If you look at user styles (or questions on support platforms) you will also notice that in spite of standardization, the inch (`in`) has not been replaced by its more correct metric counterparts. Okay, that might be due to the fact that there is no meter as unit but using smaller dimensions (`mm` and `cm`) makes more sense, also for internal accuracy reasons. That said, it is about time that we eradicate the inch or at least come up with something more metric.

So there you have it: we need some female units that correctly stay within the metric domain! In order to convince users to drop the inch the first new unit somewhat relates to it: one Edith (`es`) is the median of the widths of thumbs of BachoTeX 2023 attendees. One can argue that this is somewhat arbitrary and indeed it is. In order to get a decent value we use a discrete measurement device that groups thumbs into 15, 20, 25 and 30 mm intervals. A 10mm interval is unlikely to get many hits unless the TeX ecosystem suddenly became very easy to use and toddlers get interested in it as a game.

Rule of thumb

If we talk in terms of one Edith, we should keep in mind that at any point we can decide to re-calibrate that unit. If we end up below 25mm we probably have quite some young and/or old users in the sample set. So, in order to have a constant value, the community has to make sure that TeX (and preferably ConTeXt) usage is nicely distributed. Now, of course at BachoTeX we are quite tolerant, because also Plain and LaTeX users are sampled. Also, given that this sample of the TeX community is skewed to older users, one can wonder how that influences the initial value. It is up to the ConTeXt group to decide when and where to re-calibrate at a later moment. After all, we have to keep the narrative that ConTeXt is unstable and evolves alive, and occasionally updating a unit fits into that narrative. If you think that this kind of research is somewhat flaky, keep in mind that probably all research related to typography is kind of subjective and somewhat unreal. And BachoTeX being tagged as 'conference' adds a lot of credibility.

The Edith () makes a nice unit for margins, but it is a bit large for offsets, so we also need a female counterpart for the Knuth (). This is why, just like a centimeter () has a smaller companion in millimeter (), the Edith has a companion Tove (). In terms of points one Tove is 7.11317pt, while a Knuth is 6.43985pt. It is surely just a coincidence that the value of one Tove in points is about the age of Tove when she became aware that her dad was a ConTeXt fan. In terms of points one Edith is 71.13177pt which, ignoring the unit, comes close to the average age of those who have attended BachoTeX more than 10 times.

Originally presented at BachoTeX 2023, and to be published in the corresponding *GUST Biuletyn*.

[1] If you go back to the early days, there are even cases where you want to talk in terms of true units. Those are not affected by original TeX's magnification factor (`\mag`) but in LuaMetaTeX we dropped that factor and therefore also these true units became obsolete.

doi.org/10.47397/tb/44-2/tb137egger-edith

The implementation of these units in LuaMetaTEX is not that hard, simply because scanning for these dimensions happens in few places: when scanning dimensions, and in a Lua helper that converts a string to scaled points. At the ConTEXt meeting where we implemented the Knuth, there was some trial and error involved in order to get the right numerator and denominator. One `dk` is 422042 scaled points which brings us to a numerator 49838 and denominator 7739. Except for scaled points, the fraction gets multiplied by 65536 and the amount. Most units have numerators and denominators with weird values, although 7227 jumps out.

unit	visualized	name	num	den
bp		big point	7227	7200
cc		cicero	14856	1157
cm		centimeter	7227	254
dd		didot	1238	1157
dk		knuth	49838	7739
es		edith	9176	129
in		inch	7227	100
mm		millimeter	7227	2540
pc		pica	12	1
pt		point	1	1
sp		scaled point[a]	1	1
ts		tove	4588	645

[a] This one is not multiplied by 65536, and has been greatly enlarged to be visible here.

When you consider these numbers it is good to realize that internally the engine uses a 32 bit number, split into two halves. There is a maximum, 16383.99998pt, so that (intermediate) calculations don't overflow. The last digit of what TEX reports when it computes a dimension as points is to be taken with a grain of salt. Here is how the Edith and Tove compare to their metric counterparts:

2.5cm	4661699	71.13188pt
2.5mm	466169	7.11317pt
1es	4661692	71.13177pt
1ts	466169	7.11317pt

In case you wonder if checking for yet another unit has drawbacks in terms of performance, we can guarantee LMTX users that they won't notice a performance hit. Even with these additional units the engine quite likely beats its predecessors in scanning units. And the impact on the code base is less than 20 short lines of trivial code so that goes unnoticed as well.

Calibration

In order to conduct the calibration we need a reliable measurement device and here we got lucky. The ConTEXt community has some unique craftsmanship amongst its members and Willi Egger made us a robust sampling device that can compete with those used by the ones that the International Organization for Standards uses: the Edithorial.

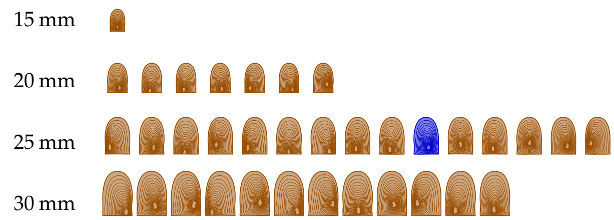


Figure 1: Results from thumb measurements at BachoTEX, with the median thumb marked in blue (grayscaled for print).

In addition to that, the ConTEXt Math Society, indeed the same one that brings you all these nice new math capabilities in LuaMetaTEX, provided the necessary statistical and mathematical underpinning to make the Edith and Tove believable units. So here are some more details.

We have found out that the Tove unit, 2.5 millimeters, corresponds to 7.1131744384765625 points. Let us find a decent rational approximation of this, with a small denominator. We do this by calculating the continued fraction, and we try a few steps to get something that is good enough.

We start by noting that the integer part is 7. We then use a calculator (in our case Wolfram Alpha) to compute

$$\frac{1}{7.1131744384765625 - 7} = \frac{1}{0.1131744384765625} \approx 8.835917486854523392207091816098.$$

This means that we get as a first possible choice

$$7 + \frac{1}{8} = \frac{57}{8} = 7.125\ .$$

We continue, and next note that

$$\frac{1}{8.835917486854523392207091816098 - 8} \approx 1.196290322580645161290322580645.$$

Thus, our next candidate is

$$7 + \frac{1}{8 + \frac{1}{1}} = \frac{64}{9} = 7.\overline{1}.$$

Here, the bar over the 1 indicates that 1 is repeating.

In the next step we calculate

$$\frac{1}{1.196290322580645161290322580645 - 1} \approx 5.094494658997534921939194741167.$$

The next candidate becomes

$$7 + \frac{1}{8 + \frac{1}{1 + \frac{1}{5}}} = \frac{377}{53} = 7.\overline{1132075471698}.$$

We continue, to get

$$\frac{1}{5.0944946589975349219391947411 67 - 5} \approx 10.582608695652173913043478260870.$$

The next approximant becomes

$$7 + \cfrac{1}{8 + \frac{1}{1 + \frac{1}{5 + \frac{1}{10}}}} = \frac{3834}{539}$$
$$= 7.\overline{1131725417439703153988868274582560296846 01}.$$

For the next step we have

$$\frac{1}{10.582608695652173913043478260870 - 10} \approx 1.7164179104477611940298507462 69$$

so the next approximant becomes

$$7 + \cfrac{1}{8 + \frac{1}{1 + \frac{1}{5 + \frac{1}{10 + \frac{1}{1}}}}} = \frac{4211}{592} = 7.1131\overline{756}.$$

Since this one has such a nice short repeating set of decimals, we fell for it, and quit here. The next approximants would be

$$\frac{12256}{1723}, \frac{20301}{2854}, \frac{28346}{3985}, \frac{48647}{6839}, \frac{466169}{65536},$$

where the last one exactly equals what we started with, 7.1131744384765625. Before we continue, we mention that $[7; 8, 1, 5, 10, 1, 1, 2, 1, 1, 9]$ is a more compact way to write the continued fraction above.

One could perhaps first think that multiplying the rational number by 10 would yield a very similar continued fraction, but that is not the case. In fact, the continued fraction for 71.131744384765625 is given by $[71; 7, 1, 1, 2, 3, 1, 3, 1, 3, 1, 3, 2]$. This put us in a bit of an awkward situation. Do we want a nice approximation for the true value, or do we prefer to have `es` to be exactly 10 times as large as `ts`? If we go for the latter, we could take 42110/592. We calculated the approximants though, and got

$$\frac{489}{7}, \frac{569}{8}, \frac{1067}{15}, \frac{2703}{38}, \frac{9176}{129}, \frac{11879}{167}, \frac{44813}{630},$$
$$\frac{56692}{692}, \frac{214889}{3021}, \frac{271581}{3818}, \frac{1029532}{14475}, \frac{2330845}{32768}.$$

When we saw this, it was irresistible to define `es` as

$$\frac{9176}{129} = 71.\overline{131782945736434108527}$$

and then to define `ts` as

$$\frac{9176}{1290} = \frac{4588}{645} = 7.1\overline{131782945736434108527}.$$

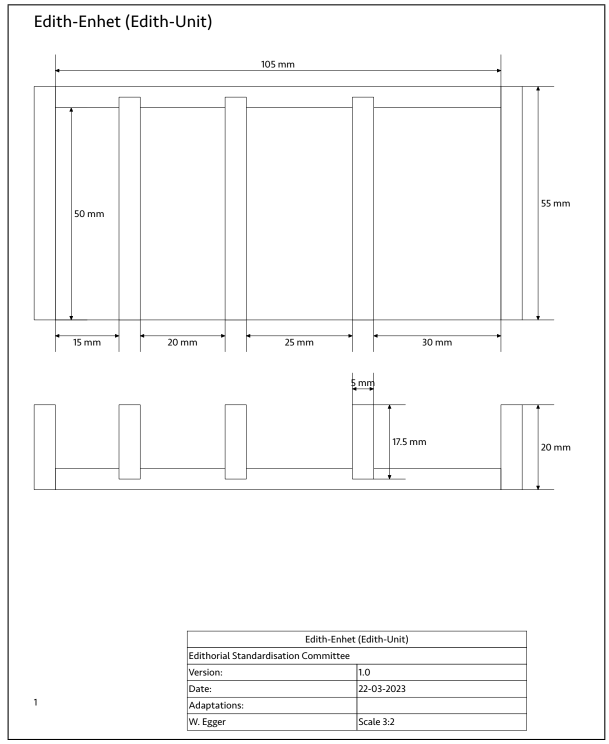


Figure 2: A TEX-community-worthy edithorial for measuring the Edith.

The edithorial device

The design of the edithorial also involved some research. Of course there was some discussion about the right way to sample thumbs and those who have attended BachoTEX and ConTEXt meetings will not be surprised that Willi is responsible for this. He presented us with a drawing (figure 2) that we immediately agreed upon.

Willi then sat down and made a prototype (figure 3) in order to see if sampling would work out. Knowing that the device would be stored under harsh conditions in the university city of Lund in Sweden, it had to be sturdy Polish oak and after being brought to precision it underwent first an iron acetate treatment and after that a furniture oil (tung oil) treatment as can be seen in figure 4. Even with TEX being digital we cannot get around physical devices for measuring digits. And with TEX operating in nanometers we have to fit in.

Some double checking

There is one question we have to answer before we dare to use the Edith (`es`) and Tove (`ts`) as offsets next to a Knuth (`dk`) and that is: in what box does Don's thumb fit? After all, we need to assign some more weight to his thumb. On the Internet you can find images of Don Knuth sitting behind an organ

Figure 3: The prototype of the edithorial.

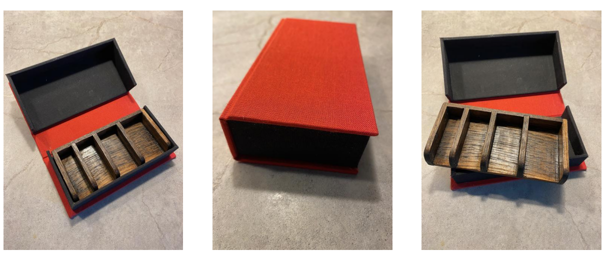


Figure 4: The reference edithorial with protective cover.

Figure 5: The 2019 lecture: Pi and *The Art of Computer Programming*.

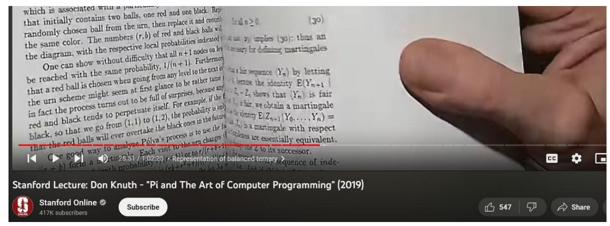


Figure 6: The 2014 lecture: (3/2)-ary Trees

but for reasons of copyright we cannot show these, but one thing we can be sure of is that his thumb is not wider than a key of that instrument, because according to the Wikipedia page `Musical_keyboard`:[2]

> Over the last three hundred years, the octave span distance found on historical keyboard instruments (organs, virginals, clavichords, harpsichords, and pianos) has ranged from as little as 125 mm (4.9 in) to as much as 170 mm (6.7 in). Modern piano keyboards ordinarily have an octave span of 164–165 mm (6.5–6.5 in), resulting in the width of black keys averaging 13.7 mm (0.54 in) and white keys about 23.5 mm (0.93 in) at the base, disregarding space between keys.

This definitely keeps Don's thumb out of the 30mm bucket. When we zoom into these images it seems also unlikely that the thumb will go to the 20mm bucket, but in the end the only one who can answer this is Don Knuth himself. And because he's behind an email firewall we don't dare to ask him. So more research was needed and after a brainstorm session we decided to rely on a public visual that any TeX user should be familiar with: the yearly Christmas lectures (figures 5, 6).

And because we know which books the thumb is on, we can calculate the bucket by comparing the dimensions: on one case we use the paper size as a reference, on the other case we use the interline spacing of the book as reference!

In Figure 7 we show a close-up of the thumb and the page. We have divided the image into a 100×100 grid, but the aspect ratio of the image is $3:2$, so we need to compensate for that. We estimate that the interline space of the text is 8 grid lines high, while the diagonal line measuring the width of the thumb is 12 grid lines wide and 42 grid lines high. This means that the thumb-interline space quotient is given by

$$\frac{\sqrt{(12 \cdot 3/2)^2 + 42^2}}{8} \approx 5.71.$$

Next, we need to know what interline space is used. We should probably know this by heart, but as we do not, we instead downloaded one of the prefascicles of *TAOCP* volume 4. We cut out a square with sides of 5 cm, and added again a 100×100 grid.

[2] Notice how metric measures win over inches here!

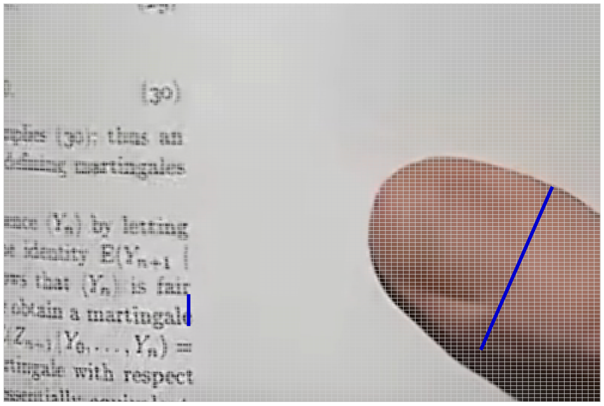
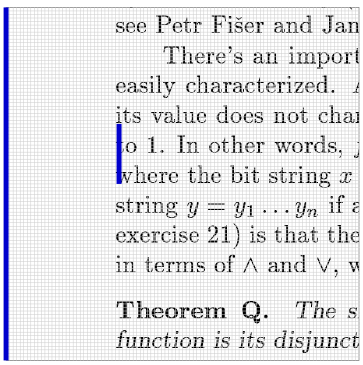


Figure 7: Close-up of Don Knuth's thumb.



see Petr Fišer and Jan
There's an import
easily characterized.
its value does not cha
o 1. In other words,
where the bit string x
string $y = y_1 \dots y_n$ if
exercise 21) is that the
in terms of ∧ and ∨, w

Theorem Q. *The s*
function is its disjunct

We measured the height of two lines and got in return 17 grid lines. This means that the interline space is given by

$$5 \times \frac{17}{2 \cdot 100} \text{ centimeter} = 0.425 \text{ centimeter} .$$

As a result we estimate that Don Knuth's thumb has the size

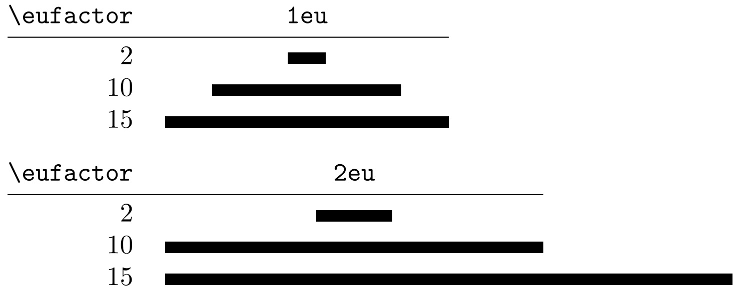
$$5.71 \times 0.425 \text{ centimeter} \approx 2.43 \text{ centimeter} .$$

If we're right about all this then the Edith will not be influenced by the grand wizard's thumb, so the well-calibrated (derived) Tove cannot be discarded for offsets as being less accurate (and stable) as the Knuth.

A modern relative unit

Since TEX showed up, a lot has changed when it comes to computers: the computers considered powerful in the early days now fit in your pocket. One disadvantage of these portable devices is that they have a variety of display sizes. A document can easily be generated again, adapting the layout to all these devices is a bit of a pain.

This is why we introduce a new dynamic unit, the `eu` or the European Unit, but one that can be changed by setting an internal register, `\eufactor`. Because that defaults to 10, one `eu` starts out as one `es`. A nice coincidence is that one can also read it as Edith's Unit.



We can set the factor in Tove steps between 1 and 50 so that we retain a reasonable accuracy. So, this relative unit stresses the sisterhood of these two new units because `1eu` is `10ts` and `1es`. This unit might also come in handy when writing manuals so you can bet that we will use it.

These units are modern in another way too. The popular game MineCraft has its own unit, a block, as (for instance) discussed on `minecraft.fandom.com/wiki/Tutorials/Units_of_measure`. For those using inches, one inch is 0.0254 blocks, so one block makes 39.3700787in. For those using metric system one cm equals 0.01 blocks or 0.16 pixels and therefore one block makes 40cm. These 40cm are 16 Ediths which means that the Edith is also a good introduction in the hexadecimal numbering system. Unfortunately LEGO bricks are defined in inches so there the inchers still have the edge. But Edith and Tove have an advantage in MineCraft, which is confirmed by observation. Just like some of TEX's units are actually defined using the inch paradigm, we could add units like `mb` for MineCraft Block being 16 Ediths. After all, implementing extra units is trivial in LuaMetaTEX. Let us know what you think.

How about MetaPost?

We not only have to deal with TEX but also with MetaPost, so from now on Metafun will also provide these units, which we can then use to properly draw thumbs as in figure 8.

While checking other units in Metafun we were reminded that they are there given as floats and not as fractions. We were amused to see

```
mm :=  2.83464 ;
cm := 28.34645 ;
```

which means that a `mm` is not exactly one tenth of a `cm`, and also that the rounding has been done by the even/odd rounding off rule. We decided to define

```
es := 71.13174 ;
ts :=  7.11317 ;
```

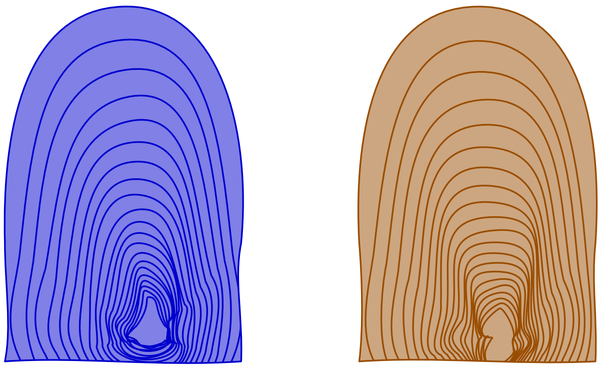


Figure 8: One can sign documents with these calibrated thumbs.

Wrapping up

In this article we discussed a few additional units that have been added to LuaMetaTEX. We've carefully chosen some names that not only compensate the male dominance in unit names, but also have a modern and fresh ring. The units are of course metric. The Edith (`es`) replaces the deprecated inch (`in`) and the Tove (`ts`) can be used for offsets as alternative to the Knuth (`dk`) that of course we will keep using alongside. The units are calibrated using an edithorial of which there exists a unique reference measurement piece. The standard has been established at the 2023 BachoTEX meeting and might be recalibrated at a future ConTEXt meeting when a new generation of users thinks that is needed.

Appendix: Overflow

When you enter a dimension in TEX and it is larger than 16383.99998pt or 1073741823 scaled points, an error message is shown and when you ask for help, that contains the sentence "I can't work with sizes bigger than about 19 feet". There is no `ft` unit in TEX, so the user has to do some conversion, maybe taking ones own foot into account.

Just like we had to adapt the error message issued when an unknown unit is used, we decided make the overflow message a bit more detailed. For that, we introduced the Theodore, where that unit is to the Edith what the foot is to the inch, and with one Theodore being five Edith. We now report this:

> I can't work with sizes bigger than about 19 feet (45 Theodores as of 2023), 575 centimeters, 2300 Toves, 230 Ediths or 16383 points.

So how did we come to this one? At the BachoTEX meeting the 18-month-old, always good-humored, Theodore was running around in the conference room and his little feet were carefully measured by his father Arthur Rosendahl (the self-appointed High Commissioner of Hyphenation and upcoming TUG president). Because the 19 feet are also an approximation, we rounded the Theodore to five Ediths. In addition we mention a few more maxima, so that the user gets a better impression how large TEX can go.

Mojca Miklavec, who gets her feet dirty by managing the binary build farm on the context garden, proposed a `th` unit but as there is no `ft` we didn't come to a conclusion yet. Although that unit would make a good default for text width, just like an `es` makes perfect left margin, and a `ts` a nice offset around framed content ...

◇ Willi Egger
BOEDE

◇ Hans Hagen
Pragma ADE

◇ Edith Sundqvist
Palettskolan
Division of Miró

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University

Producing different forms of output from XML via ConTEXt

Thomas A. Schmitz

This paper will describe the workflow that I have been using for about fifteen years now at Bonn University. I am a professor of ancient Greek and give a lecture course of about 90 minutes (i.e., 2 × 45 minutes) every week during the term. Over the course of these fifteen years, I have changed and adapted my workflow and the tools I use: after experimenting with LATEX, I used ConTEXt first to produce my lecture notes, then to produce the slides as well. After a while, it became clear that it would be beneficial to produce all material from one and only one source file. It turned out that XML was the best input format for this: it is easier to reuse for other purposes, and it allows output in different formats because of the way it is processed in ConTEXt.

Since a picture says more than a thousand words, the easiest way of demonstrating what "different forms of output" means is showing a few examples.

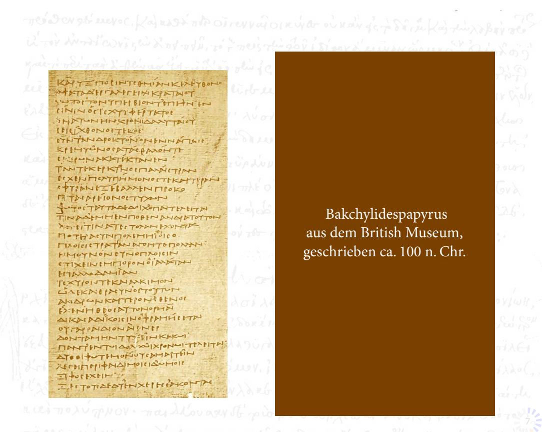
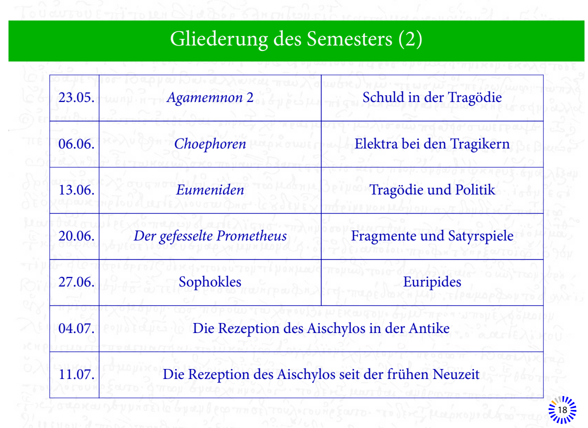


Figure 1: Slide with picture

The most important format is of course the slides that I show in my lectures. They should help students understand and master the subject; they contain different forms of material, such as pictures (figure 1 shows a slide with a picture of a papyrus fragment).



23.05.	*Agamemnon 2*	Schuld in der Tragödie
06.06.	*Choephoren*	Elektra bei den Tragikern
13.06.	*Eumeniden*	Tragödie und Politik
20.06.	*Der gefesselte Prometheus*	Fragmente und Satyrspiele
27.06.	Sophokles	Euripides
04.07.	Die Rezeption des Aischylos in der Antike	
11.07.	Die Rezeption des Aischylos seit der frühen Neuzeit	

Figure 2: Slide with table

The slide in figure 2 contains a table, another frequent format; but most of my slides contain texts and translations, like the one shown in figure 3; I teach a philological discipline, after all.

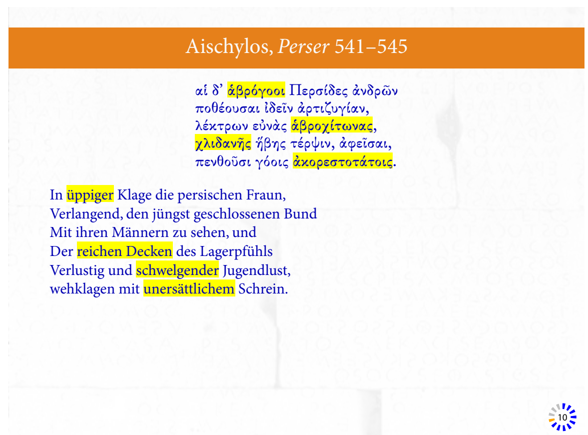
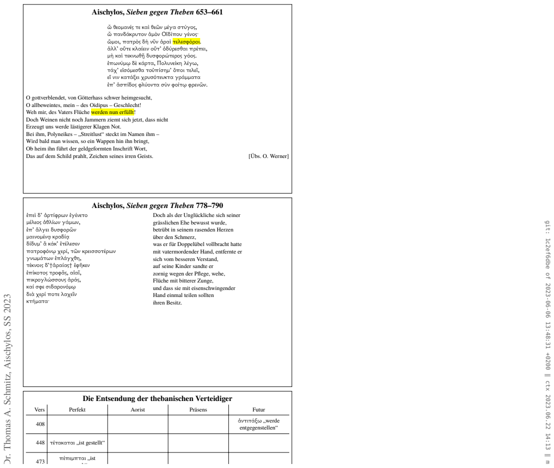


Figure 3: Slide with text

These slides have been produced with the ConTEXt module `simpleslides` that Aditya Mahajan and myself wrote many years ago. It offers a number of visually appealing styles for such presentations, and of course, it is free and open software and can be downloaded from the ConTEXt garden.

I had always made these slides available to students, originally by uploading them to my departmental website. Then, some years ago, students asked me to provide them in a format that would be easier for them to print and reuse.

doi.org/10.47397/tb/44-2/tb137schmitz-xmlcontext


Aischylos, *Sieben gegen Theben* 653–661

ὦ θεομανές τε καὶ θεῶν μέγα στύγος,
ὦ πανδάκρυτον ἁμὸν Οἰδίπου γένος·
ὤμοι, πατρὸς δὴ νῦν ἀραὶ τελεσφόροι.
ἀλλ’ οὔτε κλαίειν οὔτ’ ὀδύρεσθαι πρέπει,
μὴ καὶ τεκνωθῇ δυσφορώτερος γόος.
ἐπωνύμῳ δὲ κάρτα, Πολυνείκη λέγω,
τάχ’ εἰσόμεσθα τοὐπίσημ’ ὅποι τελεῖ,
εἴ νιν κατάξει χρυσότευκτα γράμματα
ἐπ’ ἀσπίδος φλύοντα σὺν φοίτῳ φρενῶν.

O gottverblendet, von Götterhass schwer heimgesucht,
O allbeweintes, mein – des Oidipus – Geschlecht!
Weh mir, des Vaters Flüche werden nun erfüllt!
Doch Weinen nicht noch Jammern ziemt sich jetzt, dass nicht
Erzeugt uns werde lästigerer Klagen Not.
Bei ihm, Polyneikes – „Streitlust“ steckt im Namen ihm –
Wird bald man wissen, so ein Wappen hin ihn bringt,
Ob heim ihn führt der geldgeformten Inschrift Wort,
Das auf dem Schild prahlt, Zeichen seines irren Geists. [Übs. O. Werner]

Aischylos, *Sieben gegen Theben* 778–790

ἐπεὶ δ’ ἀρτίφρων ἐγένετο	Doch als der Unglückliche sich seiner
μέλεος ἀθλίων γάμων,	grässlichen Ehe bewusst wurde,
ἐπ’ ἄλγει δυσφορῶν	betrübt in seinem rasenden Herzen
μαινομένᾳ κραδίᾳ	über den Schmerz,
δίδυμ’ ἃ κάκ’ ἐτέλεσεν	was er für Doppelübel vollbracht hatte
πατροφόνῳ χερί, τῶν κρεισσοτέρων	mit vatermordender Hand, entfernte er
γνωμάτων ἐπλάγχθη,	sich vom besseren Verstand,
τέκνοις δ’†ἀραίας† ἐφῆκεν	auf seine Kinder sandte er
ἐπίκοτος τροφᾶς, αἰαῖ,	zornig wegen der Pflege, wehe,
πικρογλώσσους ἀράς,	Flüche mit bitterer Zunge,
καί σφε σιδαρονόμῳ	und dass sie mit eisenschwingender
διὰ χερί ποτε λαχεῖν	Hand einmal teilen sollten
κτήματα·	ihren Besitz.

Die Entsendung der thebanischen Verteidiger

Vers	Perfekt	Aorist	Präsens	Futur
408				ἀντιτάξω „werde entgegenstellen“
448	τέτακται „ist gestellt“			
473	πέπεμπται „ist			

Dr. Thomas A. Schmitz, Aischylos, SS 2023

git: 1c2ef6dbe of 2023-06-06 13:48:31 +0200 ‖ ctx 2023.06.22 14:13 ‖ m

Figure 4: Printout for students

The format shown in figure 4 is a design that the students themselves suggested: on an A4 sheet, the left side has a very simplified and scaled down view of the slides, minus all the decorations, backgrounds, and fancy effects, but still keeping all the important elements such as images, tables, and text; the right column is blank so students can take notes next to the slides and thus be sure what their notes refer to.

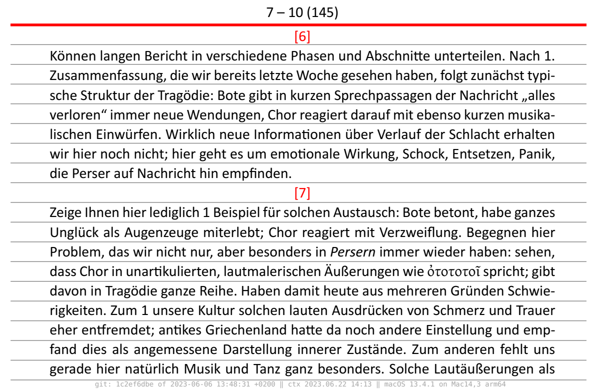

[21] *The Cambridge Companion to Greek Tragedy*, hrsg. von Patricia E. Easterling, Cambridge (Engl.) 1997.
[22] *A Companion to Greek Tragedy*, hrsg. von Justina Gregory, Oxford 2005.
[23] Dale, Ann Marjory: *Collected Papers*, Cambridge (Engl.) 1969.
[24] Dale, Ann Marjory: „Seen and Unseen on the Greek Stage: a Study in Scenic Conventions“, *Wiener Studien* 69 (1956) 96–106 (Nachdruck in Dale [23], 119–29).
[25] *Dionysus Since 69. Greek Tragedy at the Dawn of the Third Millenium*, hrsg. von Edith Hall, Fiona Macintosh und Amanda Wrigley, Oxford 2005.
[26] Fritz, Kurt von: *Antike und moderne Tragödie. Neun Abhandlungen*, Berlin 1962.
[27] Goldhill, Simon: *Reading Greek Tragedy*, Cambridge (Engl.) 1986.
[28] Gould, John: „Dramatic Character and ‘Human intelligibility’ in Greek Tragedy“, *Proceedings of the Cambridge Philological Society* 24 (1978) 43–67 (Nachdruck in Gould [29] 78–111).
[29] Gould, John: *Myth, Ritual, Memory, and Exchange. Essays in Greek Literature and Culture*, Oxford 2001.
[30] *Greek Tragedy and the Historian*, hrsg. von Christopher Pelling, Oxford 1997.
[31] *Das griechische Drama*, hrsg. von Gustav Adolf Seeck, Darmstadt 1979.
[32] Griffin, Jasper: „The Social Function of Attic Tragedy“, *Classical Quarterly* 48 (1998) 39–61.
[33] Henrichs, Albert: „‚Why Should I Dance?‘: Choral Self-Referentiality in Greek Tragedy“, *Arion* 3 (1994/95) 56–111.
[34] Henrichs, Albert: „Loss of Self, Suffering, Violence: the Modern View of Dionysus from Nietzsche to Girard“, *Harvard Studies in Classical Philology* 88 (1984) 205–40.
[35] Latacz, Joachim: *Einführung in die griechische Tragödie* (UTB 1745), Göttingen 1993.
[36] Lesky, Albin: *Die tragische Dichtung der Hellenen* (Studienhefte zur Altertumswissenschaft 2), Göttingen ³1972.
[37] Mastronarde, Donald J.: „Actors on High: The Skene Roof, the Crane, and the Gods in Attic Drama“, *Classical Antiquity* 9 (1990) 247–94.
[38] Meier, Christian: *Die politische Kunst der griechischen Tragödie*, München 1988.
[39] Melchinger, Siegfried: *Das Theater der Tragödie. Aischylos, Sophokles, Euripides auf der Bühne ihrer Zeit*, München 1974 (Nachdruck 1990).
[40] *Nothing to Do with Dionysos? Athenian Drama in Its Social Context*, hrsg. von John J. Winkler und Froma I. Zeitlin, Princeton 1990.
[41] *Oxford Readings in Greek Tragedy*, hrsg. von Erich Segal, Oxford 1983.
[42] Pickard-Cambridge, Arthur W.: *The Dramatic Festivals of Athens*, Oxford ²1968 (revised with a new Supplement (1988) by John Gould and D. M. Lewis).
[43] Pickard-Cambridge, Arthur W.: *The Theatre of Dionysos in Athens*, Oxford 1946.
[44] Pohlenz, Max: *Die griechische Tragödie*, Göttingen ²1954.
[45] Rehm, Rush: *Greek Tragic Theatre*, London 1992.
[46] Rehm, Rush: *Radical Theatre: Greek Tragedy and the Modern World*, London 2003.

Thomas A. Schmitz

Figure 5: Bibliography

Finally, I produce handouts from my source file. For regular lecture courses, these contain just the bibliography; for invited lectures or conference talks, they may also contain passages from original texts, translations, or excerpts from scholarly literature. As you can see in figure 5, I am very fond of numbered bibliographical styles because they make it so much easier and more efficient to refer to single items on the list.

What we have seen so far is, if you like, the client-facing side of my business, but of course the material I produce for myself, i.e., the notes for my lectures, is as important as these documents.


7 – 10 (145)

[6]

Können langen Bericht in verschiedene Phasen und Abschnitte unterteilen. Nach 1. Zusammenfassung, die wir bereits letzte Woche gesehen haben, folgt zunächst typische Struktur der Tragödie: Bote gibt in kurzen Sprechpassagen der Nachricht „alles verloren“ immer neue Wendungen, Chor reagiert darauf mit ebenso kurzen musikalischen Einwürfen. Wirklich neue Informationen über Verlauf der Schlacht erhalten wir hier noch nicht; hier geht es um emotionale Wirkung, Schock, Entsetzen, Panik, die Perser auf Nachricht hin empfinden.

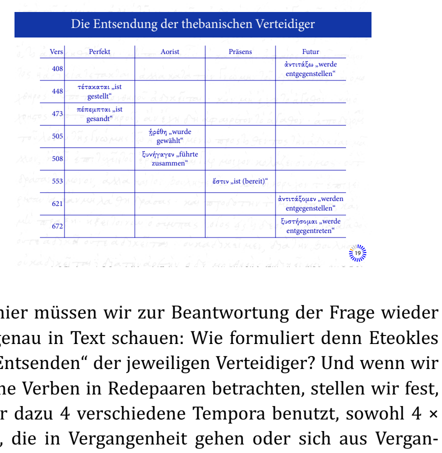
[7]

Zeige Ihnen hier lediglich 1 Beispiel für solchen Austausch: Bote betont, habe ganzes Unglück als Augenzeuge miterlebt; Chor reagiert mit Verzweiflung. Begegnen hier Problem, das wir nicht nur, aber besonders in *Persern* immer wieder haben: sehen, dass Chor in unartikulierten, lautmalerischen Äußerungen wie ὀτοτοτοῖ spricht; gibt davon in Tragödie ganze Reihe. Haben damit heute aus mehreren Gründen Schwierigkeiten. Zum 1 unsere Kultur solchen lauten Ausdrücken von Schmerz und Trauer eher entfremdet; antikes Griechenland hatte da noch andere Einstellung und empfand dies als angemessene Darstellung innerer Zustände. Zum anderen fehlt uns gerade hier natürlich Musik und Tanz ganz besonders. Solche Lautäußerungen als

git: 1c2ef6dbe of 2023-06-06 13:48:31 +0200 ‖ ctx 2023.06.22 14:13 ‖ macOS 13.4.1 on Mac14,3 arm64

Figure 6: Manuscript as “index cards”

Figure 6 shows a no-nonsense, very basic format for my manuscript: the locations where I have to change slides are simply indicated by red numbers (grayscaled for print); the slides themselves are not rendered. This is wonderfully readable on small devices such as phones or tablets, and it has the benefit that the amount of text on these “index cards” is pretty standardized, so I know exactly how many of these pages it takes to fill a 45-minute presentation; this helps me preparing my lectures.


Die Entsendung der thebanischen Verteidiger

Vers	Perfekt	Aorist	Präsens	Futur
408				ἀντιτάξω „werde entgegenstellen“
448	τέτακται „ist gestellt“			
473	πέπεμπται „ist gesandt“			
505		ᾑρέθη „wurde gewählt“		
508		ξυνήγαγεν „führte zusammen“		
553			ἔστιν „ist (bereit)“	
621				ἀντιτάξομεν „werden entgegenstellen“
672				ξυστήσομαι „werde entgegentreten“

19

Auch hier müssen wir zur Beantwortung der Frage wieder ganz genau in Text schauen: Wie formuliert denn Eteokles sein „Entsenden“ der jeweiligen Verteidiger? Und wenn wir einzelne Verben in Redepaaren betrachten, stellen wir fest, dass er dazu 4 verschiedene Tempora benutzt, sowohl 4 × solche, die in Vergangenheit gehen oder sich aus Vergan-

Figure 7: Manuscript with view of slides

In general, however, I prefer to see the slides on my tablet when I give the lecture, in exactly the form that the students see them on the big screen behind me so I have text and translation or other elements before me when I talk about them; hence I produce the format shown in figure 7 that has the slide on top of the page and then the text of the lecture below.

All these different formats and outputs are produced from one common source file written in XML, `lecture.xml`. This file is compiled with ConTEXt. I will provide just a few basic pieces of information about the way ConTEXt processes XML; for readers who want to learn about this topic in depth, there is the document *Dealing with XML in ConTEXt MkIV* in the ConTEXt distribution that contains all the details. Like most things in ConTEXt, processing XML is done via setups, which are usually collected in a special "environment" file.

```
\startxmlsetups xml:presentation_setups
   \xmlsetsetup {#1} {*} {-}
   \xmlsetsetup {#1} {lecture|
                    presentation} {xml:*}
\stopxmlsetups
\xmlregistersetup {xml:presentation_setups}
```

The code above shows the beginning of such an environment file. Setups are collected in `start/stop` pairs; in this case, they are the setups for the entire document. Line 2 is important: it tells ConTEXt to drop everything and not typeset any XML element unless it is explicitly mentioned in the following list and has its proper setup; if this line is not present, ConTEXt will use some heuristics to retrieve and typeset at least the text from all elements. And then follows this list of elements, hierarchical from the root element down to all the other subelements.

```
\startxmlsetups xml:presentation
  \xmlflush {#1}
  \par
\stopxmlsetups

\startxmlsetups xml:presentation
  \color [red] {\xmlflush {#1}}
  \page
\stopxmlsetups

\startxmlsetups xml:presentation
  \startsection [title=\xmlatt {#1} {title}]
     \xmlflush {#1}
  \stopsection
\stopxmlsetups
```

After collecting the elements that we want processed, we create setups for every single one of them; they all live in the `xml:` namespace. Just to provide an idea of what these setups look like, we see above some examples of how an element with the name `presentation` could be processed. The first setup is the most basic one: the command `\xmlflush` on l.2 takes the content of the current element (in our case the element `presentation`; the `#1` stands for the content of the current XML node) and puts its content into the TEX stream, where text will be typeset, subelements will be processed according to the rules defined in their own setups, etc. After every element, ConTEXt will insert a paragraph break. Of course, you can apply all the usual ConTEXt commands to the content of the element, as the second example shows: here, we flush the content inside a command that will apply the color red to all text, and we insert a page break at the end of the element. The third setup assumes that the element `presentation` has an attribute `title`, which can be retrieved and typeset with the command `\xmlatt` and the name of the attribute; here, we use this attribute as the title of a ConTEXt section in our document, then flush its content inside this `start/stop` pair.

```
\startxmlsetups xml:framed
  \xmlfunction {#1} {framed}
\stopxmlsetups
```

It is also possible to process the content of our XML in Lua, as the code above shows. In this case, we hand the content of the element `framed` over to a Lua function `xml.functions.framed`, which we define like this:

```
function xml.functions.framed (t)
  if tex.modes ["print"] then
    scale_factor = 0.5
  else
    scale_factor = 1
  end

  context.framed (
    { width = scale_factor *
       tex.dimen ["textwidth"] },
        function () lxml.flush (t) end )
end
```

Within this Lua function, we have access both to the power of the Lua programming language and to all ConTEXt commands, and this makes some tasks somewhat easier. The example here is quite basic to provide an idea of what can be done: first, we define a numerical value `scale_factor` that depends on the ConTEXt mode: if processing takes place in mode `print` (this is the mode that produces the handout where slides are typeset with a reduced width), this value is set to `0.5`, in all other modes it is set to `1`. We then use this value in calculating the width of our frame: we multiply it with the dimension `textwidth`. The result will be that this frame will be spread across the entire page in other modes and across half of the page in mode `print`. Within this frame, we then flush the content of our element. The last line of our code shows a difficulty: some ConTEXt commands need to be finished before typesetting can begin; this is done with the somewhat clunky `function ... end` structure in this line. If you want to learn more about these details, you can have a look at the manual *ConTEXt Lua documents*, which is also part of the ConTEXt distribution.

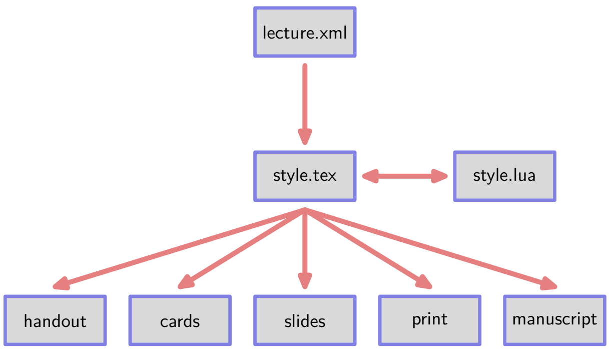


Figure 8: Schema of the workflow

This, then, is a short summary of the way in which ConTEXt processes XML. Knowing these general rules, we now understand the workflow that I have created for my lecture; you find a schematic representation in figure 8: in order to compile the `xml` file `lecture.xml`, we need a ConTEXt style file `style.tex`, which is connected with a Lua file `style.lua`, where Lua commands are defined; both files control the typesetting process. I will describe some of the salient features that allow me to derive different forms of output from the same XML file.

```
<lecture language="en" style="BigNumber">
  <presentation date="23_05_16">
    <title>The <emph>Persae</emph></title>
    <content>
      <p>Text ... </p>
      <slide> ... </slide>
    </content>
  </presentation>
  <presentation date="23_05_23">
    <title>The <emph>Supplices</emph></title>
    <content>
      <p>More Text ... </p>
      <slide> ... </slide>
    </content>
  </presentation>
  <presentation date="23_05_30">
    <title>The <emph>Agamemnon</emph></title>
    <content>
      <p>And still more Text ... </p>
      <slide> ... </slide>
    </content>
  </presentation>
</lecture>
```

We begin by taking a brief look at the main structure of my `lecture.xml` file, which is summarized in the code sample above. The text for the entire lecture course of one semester is contained in this XML file, which will grow to around 30,000 lines over the course of the teaching term. The structure is quite simple: the root element `lecture` consists of a number of individual `presentations`; each one of them has a an attribute `date` and subelements `title` and `content`; this content consists of the text (i.e., my lecture notes) and the slides that will be shown on the screen.

When we compile this document, it is obviously efficient to have all individual presentations in one document: this way, it is easy to create cross-references, a bibliography with consistent numbering, and to move content around. However, when we want to compile the slides for an individual lecture, we do not want all the slides for all presentations included in the resulting PDF file (which would become huge in size because of included images, etc.), but just the ones for the current lecture. This is why every individual lecture has a `date` tag. We will use this attribute to filter individual lectures by making use of a feature of ConTEXt that is called "modes". Modes provide a means for conditional typesetting; they allow us to include or exclude parts of our source into the typesetting process.

```
\doifmode {presentation} {
 \doifmode {\xmlatt {#1} {date}} {
   \setupTitle [title = {\xmltext {#1} {title}}]
   \xmltext {#1} {content} \page
 }
}
```

The code above shows how this works. We first define a block that will only be processed when ConTEXt is in mode `presentation`; this is what the first `\doifmode` line does. Inside this block, we nest a second mode, which is set from the `date` attribute of our `presentation` element: ConTEXt will execute the following code only when its mode is equal to this `date` attribute. It will skip over all other `presentation` elements and only typeset the one that corresponds to its mode (i.e., its date) if and only if it is in `presentation` mode; if it is in any other mode (typesetting the manuscript or the bibliography), it will process all `presentation` elements.

We pass these modes to ConTEXt when we call it from the command line. So in our case, the call would be `context --environment=lecture-style --mode=presentation,23_05_23`, and ConTEXt will process only the `presentation` element with the `date` attribute of 23_05_23.

```
\startxmlsetups xml:slide
  \startmode [presentation]
    \xmlfunction {#1} {presentation_slide}
  \stopmode
  \startmode [cards]
    \incrementcounter [slide_number]
```

```
    \midaligned {\color [red]
      {[{\rawcounter [slide_number]}]}}
  \stopmode
  \startmode [combined]
    \xmlfunction {#1} {combined_slide}
  \stopxmlsetups
\stopxmlsetups
```

Here is another examples where the modes mechanism allows conditional typesetting and filtering of content. For the element `slide`, we define three different setups, dependent on the mode we are using. When we are in mode `presentation`, ConTEXt will pass the entire content of this element on to the Lua function `presentation_slide`, where we take care of the different subelements, such as including pictures, typesetting tables and text, etc.

However, when we are in mode `cards` and want to produce our index cards, something else happens: ConTEXt will simply increase a special counter `slide_number` for slides by 1 and typeset the result on its own line; we have already seen the result in figure 6. This way, every place where we need to advance our slides is clearly marked in our manuscript; we will not run the risk of forgetting that a new slide was supposed to come up.

When we are in `combined` mode, the element content is passed to a Lua function `combined_slide` that is defined in our `style.lua` file. We have seen the general outlines of how such Lua functions work; here we find another example of things that may be easier to handle in Lua than in TEX.

```
local i = 1
local ctx = context

function xml.functions.combined_slide (t)
  i = i + 1
  local current = xml.attribute
    (t, "../../", "tag", "")
  local textwidth = tex.dimen.textwidth
  ctx.page ()
  ctx.framed ( { width=number.todimen(textwidth)
    frame="off", align="middle",
    height="10cm" }
         function ()
         ctx.externalfigure({ "presentations/"
            .. current .. ".pdf"},
         { page=i, width="13cm" } )
         end )
       ctx.blank { "line" }
end
```

ConTEXt converts every XML element into a Lua table, and we pass this table `t` as an object to the Lua function `combined_slide`. We retrieve the date of the current presentation in the variable `current`. Since we know that the element `presentation` is “grandparent” of the element `slide`, we have to move two levels up to find this attribute; this is what the expression `xml.attribute (t, "../../", "tag", "")` does. This information will be used when we construct the object of the ConTEXt command `\externalfigure`: the slides for every single presentation are stored as PDF files in a subdirectory `presentations/`, and they are named with the value of their `date` attribute, so a presentation shown on July 15 would be named `23_07_15.pdf`. When we concatenate the strings and variables `"presentations/" .. current .. ".pdf"` in Lua, we make sure that ConTEXt will use this file as an external figure. Finally, we increase the counter `i` each time this macro is called; this counter is used to retrieve the single pages of our PDF file, so they will be shown one by one, in ascending order. Before the picture of the slide, ConTEXt inserts a page break. Again, we have seen the result of this code above in figure 7: a small image of every slide will be on top of the page or pages containing the notes to the slide.

This, then, is the general mechanism to produce different forms of output from a single XML file: we use ConTEXt modes to apply different setups to different XML elements; depending on which mode we compile with, the slides from a certain presentation or the text for the notes or the bibliography will be typeset; instead of typesetting the slides, we may increase a counter or include them as images. I hope that the general principle is clear now. I want to conclude this brief overview with two special cases that have been useful over years.

As I have shown, XML is a versatile and self-testing input format: if the XML code is valid, it should compile in ConTEXt; you do not have to worry about closing groups and nesting brackets. Nevertheless, it necessitates some work: for every detail you want to typeset in your manuscript or on your slides, you have to come up with XML code to represent it and with a setup to translate it into something ConTEXt can process.

This is worthwhile for code that you use over and over again, but it makes it more difficult to write small adjustments. And certain tasks are much easier to code in TEX rather than in XML. One example is MetaPost graphics. In theory, it would be possible to write XML code that would then be translated to MetaPost code, typeset by ConTEXt and displayed on your slides. But this would be exceedingly painful and demand lots of work. It would be preferable to have some way of simply writing MetaPost code (or even arbitrary TEX code) and have it executed when ConTEXt processes our XML file. ConTEXt has a macro `\processTEXbuffer` that does just this: it executes included TEX code and typesets the result. We use this mechanism by defining an XML element `processbuffer` and writing a setup for it; the code below shows how to do this.

```
\startxmlsetups xml:processbuffer
  \processTEXbuffer [\xmlflush {#1}]
\stopxmlsetups

\doiffileelse {presentation_buffers.tex}
       {\input presentation_buffers}
       {\relax}
```

Moreover, I have found it more efficient to collect all the different TEX buffers for my lectures in one TEX file `presentation_buffers.tex`. Within this file, I have the TEX code for my various buffers, e.g., the MetaPost code that produces figure 8 (I include a short extract only).

```
\startbuffer [workflow]
\startmode [presentation,combined]
\setupMPvariables [workflow] [a=0.5in, b=0.2in]
\stopmode
\startmode [print]
\setupMPvariables [workflow] [a=0.25in, b=0.1in]
\stopmode

\startuniqueMPgraphic{workflow}
numeric a, b ; a = \MPvar {a} ; b = \MPvar {b} ;
path p[] , q[] ;

p[1] := fullsquare xyscaled (a,b) ;
fill p[1] withcolor \MPcolor {lightgray} ;
label (textext ("slides"), center p[1]) ;
\stopuniqueMPgraphic

\uniqueMPgraphic {workflow}
\stopbuffer
```

This way, our XML file can simply call the buffers in this file inside the element `processbuffer`:

```
<slide>
  <slidecontent>
    <processbuffer>workflow</processbuffer>
  </slidecontent>
</slide>
```

Finally, typing these long ConTEXt calls with the proper `--environment` and the proper `--mode` on the command line was pretty cumbersome, so I delegated this part to a Makefile, where I specify the date of the presentation I want to typeset in a variable `day`; a short extract of this Makefile would look like this:

```
combined: lecture.xml
       context --environment=style \
         --mode=presentation,$(day) lecture.xml
       cp lecture.pdf ./presentations/$(day).pdf
       #
       context --environment=style \
         --mode=combined,$(day) lecture.xml
```

When I call this Makefile with the call `make day=23_05_17 combined`, ConTEXt will first compile in mode `presentation`, then copy the current presentation into the proper directory under the correct name, then run once again in mode `combined` and include the single slides into the lecture notes. This is fairly simple, but it saves a lot of typing over the course of a semester.

I hope this paper has raised some interest and shown the advantages of producing all our output for a lecture from one single XML file.

⋄ Thomas A. Schmitz
Institut für Klassische und Romanische Philologie
Universität Bonn
Rabinstraße 8
53111 Bonn
Germany
`thomas dot schmitz (at) uni-bonn dot de`

LaTeX News

Issue 37, June 2023 (LaTeX release 2023-06-01)

Contents

New functionality offered as part of the "LaTeX Tagged PDF" project

We have now enabled new automatic tagging functionality for additional LaTeX elements, among them most display environments, standard sectioning commands, content, figure and table listings, floats and graphics and bibliographies. This can be activated through

```
\DocumentMetadata{testphase=phase-III}
```

At this point in time tagging support is only available for a restricted set of documents, i.e., those that use one of the basic document classes (`article`, `report`, and `book`) and only use commands and environments described in Lamport's LaTeX manual.

Using other document classes or adding additional packages in the preamble may work (or may partially work) but at this stage it is not very likely, at least not for packages or classes that excessively alter internals of LaTeX.

Also note that there are still several environments and commands described in the LaTeX manual that do not have tagging support yet, notably tabulars, `tabbing`, the various math environment and a few others. They will get this support as part of `phase-III`, but some of them will be delayed until after the June release.

A prototype for math tagging (including support for the amsmath environments) is already available, but it is mainly intended for experimentation and feedback and the resulting tagging is by no means the way we envision it to be eventually. If you would like to try it out use the following line:

```
\DocumentMetadata{testphase={phase-III,math}}
```

Note that the math tagging code at this point in time will clash with packages that redefine the `$` character (which then may lead to strange errors) and that packages that use math mode for non-mathematical constructs may result in surprising output as far as tagging is concerned. Feedback on which packages fail with the code in one or another way would be appreciated.

The `latex-lab` bundle contains various (still untagged) documentation files about the new code that can be accessed with `texdoc -l latex-lab`.

Feedback is welcome! Please use `https://github.com/latex3/latex2e/discussions/1010`.

New or improved commands

Extending hooks to take arguments

Hooks have always been containers for code whose outcome was entirely dependent on the contents of the hook alone. If any type of contextual information had to be passed to the hook, it had to be done by setting some variable before the hook so that the code in the hook could use

doi.org/10.47397/tb/44-2/tb137ltnews37

that. But this is somewhat hard to keep track of, clumsy to implement, and it required the programmer to have some kind of “hook before the hook” to do that setup.

To make things a bit easier, lthooks was enhanced to support hooks with arguments. Hooks can now be declared and used with arguments, then the code added to these hooks can reference the hook’s arguments using `#1`, `#2`, etc., so now hooks can behave more like macros than like *token lists* (using expl3 terminology). Regular argument-less hooks continue to work exactly like they did before: this extension is completely compatible with older documents and packages.

To declare a hook with arguments, use

`\NewHookWithArguments` {⟨*hook*⟩} {⟨*num-args*⟩}

then, similarly, to use the code in the hook, supposing a hook declared with 2 arguments, write

`\UseHookWithArguments` {⟨*hook*⟩} {2} {⟨*arg1*⟩} {⟨*arg2*⟩}

Or, if you want to add some code to a hook that takes arguments, write

`\AddToHookWithArguments` {⟨*hook*⟩} [⟨*label*⟩] {⟨*code*⟩}

exactly like you would for regular hooks, except that the ⟨*code*⟩ can use the arguments by referencing `#1`, `#2`, etc. In this case, if you want to add an actual parameter token (`#`) to the ⟨*code*⟩, you have to double it, as usual.

Additionally, if you want to add “regular” code to a hook with arguments, you can still use `\AddToHook` — in that case `#` tokens are *not* doubled. This means that a package author can decide to add arguments to an existing hook without worrying about compatibility: `\AddToHook` will do the right thing and will not mistakenly reference the newly added arguments.

The commands `\NewReversedHookWithArguments`, `\NewMirroredHookPairWithArguments`, `\AddToHookNextWithArguments`, `\UseOneTimeHookWithArguments`, and the expl3 counterparts of the commands discussed in this section were also added. The complete documentation can be found in the lthooks documentation [2].

Generic cmd hooks with arguments: Along with the possibility of passing arguments to a regular hook as discussed above, generic `cmd` hooks can now access the arguments of the command they are patched into, using the interface described in the previous section.

For example, if you were to add some code to the `\title` command using hooks, you could access the actual title given in the argument. Thus, to write the title of the document in the terminal you could use:

```
\AddToHookWithArguments{cmd/title/before}
  {\typeout{Document title: #1}}
```

As with regular hooks, code added to a `cmd` hook using `\AddToHook` will not be able to access the command’s arguments. This means that, as with regular hooks, this change is completely backwards compatible, so previous usages of `cmd` hooks will work exactly as they did before.

Providing copy and show functions for environments

To copy a command definition we introduced `\NewCommandCopy` in 2022. This even allows you to copy commands that consist of several internal components, such as robust commands or those with a complex signature. To do the same with environments, e.g., to define the environment `myitemize` to be equivalent to `itemize`, you can now write

```
\NewEnvironmentCopy{myitemize}{itemize}
```

There are also `\Renew...` and `\Declare...`, which may be useful depending on the circumstances.

In addition, we offer a `\ShowEnvironment` command, which displays the `\begin` and `\end` code of the environment passed as an argument. E.g., `\ShowEnvironment{center}` results in the following output:

```
> \begin{center}=environment:
> ->\trivlist \centering \item \relax .
<recently read> }
l. ...\ShowEnvironment{center}
> \end{center}:
> ->\endtrivlist .
<recently read> }
l. ...\ShowEnvironment{center}
```

(github issue 963)

`\IfFileAtLeastTF`

The 2020-10-01 LaTeX release introduced the CamelCase tests `\IfClassAtLeastTF` and `\IfPackageAtLeastTF` for checking class and package dates. We have now added `\IfFileAtLeastTF` to allow the same to happen for generic files which contain a `\ProvidesFile` line.

(github issue 1015)

`\DeclareLowercaseMapping`, `\DeclareTitlecaseMapping` and `\DeclareUppercaseMapping`

The move from a case-changing approach using `\lccode` and `\uccode` data to one where information is stored by a kernel-managed structure left a gap in the ability of the user to *tune* the case changing outcomes. This has now been addressed by the addition of three commands

- `\DeclareLowercaseMapping`
- `\DeclareTitlecaseMapping`
- `\DeclareUppercaseMapping`

which can be used to customise the outcome for codepoints. This can be applied generally or to a specific locale (see also the next section). A small number of pre-defined customisations have been set up in the kernel where the outcomes for pdfTeX should be different for those from Unicode engines. For example

```
\DeclareUppercaseMapping{"01F0}{\v{J}}
```

allows J̌ to be produced in 8-bit engines: without this customisation, an error would occur as there is no precomposed J̌ in Unicode. More detail is given in `usrguide`.

(github issue 1033)

\BCPdata

Improvements in the Unicode handling for case changing have highlighted that the kernel has not to-date been locale-aware. The packages babel and polyglossia provide comprehensive locale support, but did not have an agreed unified interface to pass that information back to other code. Following discussion with the maintainers of those two bundles, the kernel now defines `\BCPdata` as a stub (so it is always defined), and babel and polyglossia will redefine it to provide the locale data. An agreed set of keywords mean that `\BCPdata` can be queried in a structured way by both the kernel and any other "consumer" packages. *(github issue 1035)*

Improve \samepage

The `\samepage` declaration sets various parameters to `10000` to prevent undesired page breaks. The `\predisplaypenalty` parameter has already by default a value of `10000`, and to save space in the past it was therefore not explicitly set. However, there are a few classes that change the parameter and as result the user might experience a page break in front of a display formula within the scope of `\samepage` when using such classes. This has now been corrected and `\predisplaypenalty` is also explicitly set to `10000`. *(github issue 1022)*

Groups in \MakeUppercase

Prior to 2022, `\MakeUppercase` and `\MakeLowercase` used a brace group around their argument so providing a scope for any declarations within the argument. This grouping has been restored (also for `\MakeTitlecase`), although the underlying L3 text case commands do not use grouping. *(github issue 1021)*

Extension of the \label command

Previously, in standard LaTeX, the `\label` command wrote a `\newlabel` declaration into the `.aux` file and stored two values in the second argument of this `\newlabel` command: `\@currentlabel`, which normally contains the state of the current counter and `\thepage` for the current page number.

The packages hyperref and nameref then patched the `\label` command to store five values instead. In addition to the above they saved `\@currentlabelname`, which normally contains the current title text and can be retrieved with `\nameref`, and `\@currentHref`, which is the name of the destination needed to create an active link. The fifth argument was only used if external references were loaded with the xr-hyper package.

Starting with this release, the number of values stored in `\newlabel` has been unified. `\label` now writes a `\newlabel` command that always contains five values in the second argument (each in a brace group): `\@currentlabel`, `\thepage`, `\@currentlabelname`, `\@currentHref`, and `\@kernel@reserved@label@data` (which is reserved for the kernel).

Additionally, a hook with the name `label` has been added. It takes one argument: the label string. Code added to the hook can refer to this argument with `#1`. The hook is executed directly before the `\label` command writes to the `.aux` file but *after* the `\@bsphack` command has done its spacing magic, and it is located *inside* a group; thus, its code only affects the write operation.

Code improvements

Performance in checking file existence

The addition of hooks, etc., to file operations had a side effect of making multiple checks that the file existed. In larger documents using many files, these file system operations caused non-trivial performance impact. We now cache the existence of files, such that these repeated filesystem calls are avoided.

doc: Handle \␣ correctly in the index

Due to some problems in the code it wasn't possible to prevent `\␣` from showing up in the index—`\DoNotIndex{\ }`, etc. had no effect. This has now been corrected. *(github issue 943)*

doc: Support the upquote package

The default quote and backquote characters in typewriter fonts are typographical quotes, e.g., the input

```
\verb/‘prog ’my input’‘/
```

is rendered as `‘prog ’my input’‘` and not as `` `prog 'my input'` `` as preferred by many programmers.

This can be adjusted, for example, with the upquote package, which results in the second output. However, for historical reasons doc had its own definition of `\verb` and `verbatim` and as a consequence the two packages did not cooperate. This has now been fixed and loading upquote together with doc has the desired effect. *(github issue 953)*

Default definition for \do

The command `\do` with its nice public name is in reality an internal command inherited from plain TeX for list processing. However, it only got a definition when `\begin{document}` was executed, with a result that a user definition in the preamble was unconditionally overwritten at this point. To properly alert the user that this command is not freely available we now make a definition in the format, so that `\newcommand` and friends produce a proper error message instead of allowing a definition that doesn't last. *(github issue 975)*

New key for filecontents

The `filecontents` environment warns on the terminal if a file gets overwritten even if that is intentional, e.g., when you write a temporary file over and over again. To make the warning less noisy in this case we added a new `nowarn` key that redirects the overwriting warning to the transcript file. We think that some record of the action is still required to help with debugging, thus it is not completely silenced. The warning that nothing gets written, because the file already exists (and the `force` key was not used), is not altered and still shows up on the terminal. *(github issue 958)*

A further hook for shipping out pages

Since October 2020 the shipout process offers a number of hooks to adjust what is happening before, during, and after the `\shipout`. For example, with the `shipout/before` hook, packages can reset code they have altered (e.g., `\catcode`s during verbatim-like processing) and with `shipout/background` and `shipout/foreground` material can be added to the pages. Details are given in [1].

However, still missing was a hook that allows a package writer to manipulate the completed page (with foreground and background attached) just before the actual shipout happens. For this we now provide the additional hook `shipout`. One use case (sometimes needed in print production) is to mirror the whole page via `\reflectbox` including all the extra data that may have been added into the fore- or background. *(github issue 920)*

Displaying release information in the .log

LaTeX displays its release information at the very beginning of the LaTeX run on the terminal, and also writes it to the transcript file if that is already opened at this point. While this is normally true, it is not the case if the LaTeX run was started passing additional TeX code on the command line, e.g.,

```
pdflatex '\PassOptionsToClass{11pt}{article}
              \input{myarticle}'
```

In this case the release information is displayed when `\PassOptionsToClass` is processed but the transcript file is only opened when the output file name is known, i.e., after `\input` has been seen, and as a result the release information is only shown on the terminal.

To account for this scenario, we now repeat the release information also at the very end of the transcript file where we can be sure that it is open and ready to receive material. *(github issue 944)*

Bug fixes

Incompatibility between doc and unicode-math

The unicode-math package alters the catcode of | but does not adjust its value for use in doc, with the result that "or" modules, i.e., $\langle A|B\rangle$ are displayed in a strange way. This is now fixed with some firstaid code that will eventually be moved into unicode-math. *(github issue 820)*

A fix for \hspace

The change to `\hspace`, done in 2020 to make it calc-aware, had the unfortunate side effect that starting a paragraph with `\hspace` would result in the execution of `\everypar` inside a group (i.e., any local changes would immediately be revoked, breaking, for example, wrapfig in that special situation). This got fixed with the 2022-11 PL1 hotfix, so was already corrected in the previous release, but is only now documented in the newsletter. *(github issue 967)*

Ensure that \cs is defined in ltxdoc

The class ltxdoc defined the command `\cs` to typeset a command name with a backslash in front. This definition was moved to the doc package itself. This meant that it was suddenly missing when reverting to the old doc package implementation via the class option `doc2`. This has now been corrected. *(github issue 981)*

Improve spacing at top of minipages

A list and several other document elements add some vertical space in front of them. However this should not happen at the beginning of a box (such as a `minipage`) and normally it doesn't, because TeX automatically drops such space at the start of a vertical list. However, if there is some invisible material, such as a `\color` command, a hyperref anchor, a `\write` or other such items, then the list is no longer empty and TeX no longer drops the vertical space.

With the new paragraph handling introduced in 2021 it is now finally possible to detect and avoid this problem and apply appropriate counter measures so that from now on the spacing will always be correct. *(github issue 989)*

A fix for \NewCommandCopy and \ShowCommand

When copying and showing definitions of (non-expandable) document commands (a.k.a. commands defined by `\NewDocumentCommand` and friends) containing empty or only `m`-type arguments, these commands were wrongly recognized as expandable ones. This is fixed in the present LaTeX release. *(github issue 1009)*

Corrections for switching math version

Some internal code improvements improve support for switching math version when nested within an outer math expression. This will improve `\boldsymbol` and `\bm` and similar commands. *(github issue 1028)*

Allow par as a filename

`\input{par}` or `\includegraphics{par}` could give spurious errors. This has been fixed by making an internal command `\long`. *(github issue 942)*

Correct setting of \endlinechar in +v arguments

In the particular case of a document command with a +v-type argument used inside `\ExplSyntaxOn/Off`, newlines would be misinterpreted as spaces because the value of `\endlinechar` was set too late. This has been fixed, and now newlines are correctly translated to "`the character ^^M`". *(github issue 876)*

Correct handling of hooks with only 'next' code

When `\AddToHookNext` was used on a not-yet-declared hook, that hook would be incorrectly identified as empty by `\ShowHook`. Also, if that hook was later declared, that 'next' code would not be executed. This has been fixed by correctly initializing the hook structure when `\AddToHookNext` is used. *(github issue 1052)*

Ignoring space after $$

Space is normally ignored after a closing `$$`, but internal LaTeX font handling code could interfere if `\eqno` was used. `\eqno` and `\leqno` have been redefined to add `\ignorespaces` after the math group. *(github issue 1059)*

Documentation improvements

Updates to the guides

When LaTeX 2_ε was released, the team provided documentation for both document authors and package/class developers in the two files `usrguide` and `clsguide`. Over time, the team have augmented these documents as new methods have been added to the kernel. However, they retained their structure as assuming familiarity with LaTeX 2.09. This meant that for new users, there was material which is no longer relevant, and less clarity than desirable regarding the approaches that are recommended today.

The two files have now been (partially) re-written, with the versions available previously now frozen as `usrguide-historic` and `clsguide-historic`. More material has been carried forward in the class/package guide than in the user guide, but both are worth a re-read by experienced LaTeX users.

Displaying the exact release dates for LaTeX

In some situations it is necessary to find out the exact release dates for older versions of the LaTeX format, for example, when you need to use different code in a package depending on the availability of a certain feature and you therefore want to use `\IfFormatAtLeastTF{`⟨*date*⟩`}` or the rather horrible construction `\@ifl@t@r\fmtversion{`⟨*date*⟩`}`, if you want to cater for formats that are older than 2020.

Or you know that your package is definitely not going to work with a format before a certain ⟨*date*⟩, in which case you could use `\NeedsTeXFormat{LaTeX2e}[`⟨*date*⟩`]` to ensure that users are alerted if their format is too old.

The big problem is knowing the exact ⟨*date*⟩ to put into such commands; in the past, that was not that easy to find. You could have looked in the file `changes.txt`, but that is hidden somewhere in your installation and if you try `texdoc -l changes.txt` you get more than thirty results and the right file is by no means the first.

Yukai Chou (`@muzimuzhi`) kindly provided a patch for this, so that we now have the exact dates for each LaTeX format listed in an easy to remember place: in `ltnews.pdf` and that file conveniently also contains all major features and changes to LaTeX over the years—one of which is most likely the reason you need the ⟨*date*⟩ in the first place.

The date is now given in parentheses in the newsletter title, thus this newsletter tells you that on 2023-06-01 the command `\NewEnvironmentCopy`, a new `shipout` hook, etc. was made available. And looking into `ltnews.pdf` you can now easily find out that the LaTeX3 programming layer was added on 2020-02-02 (because the date was so nice) and not on the first of the month. *(github issue 982)*

Fresh from the press: "The LaTeX Companion, third edition" is now in print

The third edition of *The LaTeX Companion* is now available. This is the result of five years of careful work and we hope that it will provide our readers with all the information they need to successfully navigate the LaTeX ecosystem and efficiently produce beautiful documents.

Since the publication of the last edition (2004), a lot has happened in the LaTeX world and thus a complete rewrite was necessary. All chapters have been thoroughly revised, and in many cases significantly extended, to describe new important functionality and features. More than 5,000 add-on packages have been analyzed in detail, out of which roughly 10% have been chosen for inclusion in *The LaTeX Companion*. All important aspects of these packages are described to provide the user once again with a satisfying one-stop–shop experience for the decade to come.[1]

To cover what we thought worth describing today, the book nearly doubled in size. The print edition is therefore produced as a two-volume set and sold as a bundle. Both volumes come as hardcover with ribbons to easily mark pages in the book.

To give you an idea of what is covered in the third edition you can find some excerpts at

`https://www.latex-project.org/news/2023/03/17/TLC3`

The edition is also available as an eBook (Parts I and II combined) consisting of PDF and ePub format, without DRM. Finally, the publisher offers the combination of the printed books and the digital versions at a very attractive price not available anywhere else.

Changes to packages in the tools category

multicol: Better support for CJK languages

The default minimum depth of each column in a `multicols` corresponds to the depth of a "p" in the current font. This helps to get some uniformity if rules are used between the columns and makes sense for Latin-based languages. Until now it was hard-wired, but for CJK (Chinese/Japanese/Korean) languages it is better to use a zero depth, because there all characters have the same height and depth. And even with Latin-based languages one might want to use the depth of a `\strut` or that of a parenthesis. So we now offer a way to adjust this while maintaining backward compatibility: redefine `\multicolmindepthstring` to hold whatever you want to get measured for its depth (the width is not relevant). *(github issue 698)*

multicol: Fix handling of nested environments

If `multicols` environments have been nested into each other (the inner one boxed) it could fail if the boxed environment appeared near a page break. The problem was that the output routine was called while the `\hsize` was still altered to fit the column width of the inner `multicols` — thereby messing up the placement of columns of the page. This has now been fixed. *(github issue 1002)*

References

[1] Frank Mittelbach, LaTeX Project Team: *The* `ltshipout` *documentation*. Run `texdoc ltshipout-doc` to view.

[2] Frank Mittelbach, Phelype Oleinik, LaTeX Project Team: *LaTeX's hook management*. Run `texdoc lthooks-doc` to view.

[1] Editor's note: A review of *The LaTeX Companion, Third Edition* appears in this issue of *TUGboat*, pp. 322–324.

Beamer news: 2023

samcarter, Joseph Wright

Abstract

The beamer class is used by many users all around the world to create slides for their presentations. This article will highlight some changes and new features, added over the last few years, which might be interesting to know for beamer users.

1 Introduction

The beamer class is the most widely used class to produce slides for presentations in LaTeX. It was first created by Till Tantau in 2003 with a lot of — for that time — revolutionary features. Since then the class has steadily evolved and gained even more features, while at the same time accumulating a very large user base. Some of these users have been using beamer for many years.

Thus, at the current point in beamer's life cycle, we maintainers focus our efforts on stability. We fix bugs and generally keep things from crumbling in the event of changes in the LaTeX kernel or one of the other packages that beamer depends on. Nevertheless, a couple of smaller changes have been made over the last few years which aim to improve the usability of the class for its users. In this article we would like to highlight some of the changes which we think are useful for beamer users to know about.

The section headings in the following will all include a version number at the right hand side. This is the version in which the respective change was introduced to the beamer class.

2 Transparent shadows (v3.60/64/70)

The first change is mostly cosmetic: beamer now has transparent shadows for blocks, headlines, etc. Previously, the shadow effect was created by adding a colour gradient from dark to the background colour of the frame (normally white). On frames with a background image, this resulted in strange looking halos, as can be seen in the top panel of Figure 1.

The first step to fixing this was a pull request by Andrey Paramonov for beamer v3.51. While this worked great for most engines, it caused some problems for DVI-based compilation chains and thus was ultimately reverted in the next beamer version. With the help of Ulrike Fischer, transparent shadows were finally resuscitated in v3.60 and, after an additional 10 versions of tracking down non-transparent shadows of various elements, all shadows in beamer v3.70 should now finally be transparent.

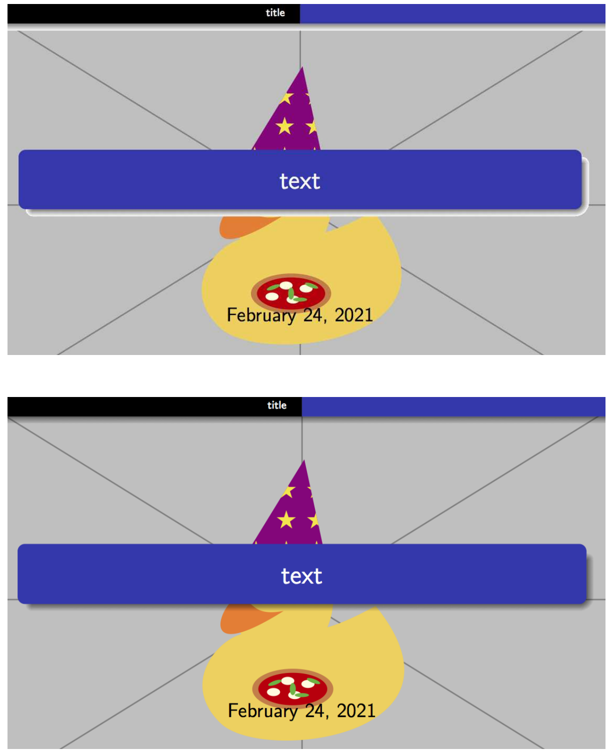


Figure 1: Comparison of non-transparent (top image) and transparent shadows (bottom image)

3 Aspect ratios (v3.65)

Surprisingly many different aspect ratios are used around the world. In the past, beamer only offered a limited set of available aspect ratios. Starting with beamer v3.65, beamer can calculate new aspect ratios on the fly. In addition to the existing options, the user can pass a two to four digit number to the `aspectratio` class option and beamer will calculate the frame geometry accordingly.

For two and four digit numbers, the number is split in the middle to obtain the width-to-height ratio of the frame; for three digit numbers, a landscape format is assumed and the number is split after the second digit:

- 2 digits: `aspectratio=23` as `2:3`
- 3 digits: `aspectratio=137` as `13:7` (always landscape orientation)
- 4 digits: `aspectratio=4310` as `43:10`

Internally, beamer uses a fixed frame height of 9.6 cm for all newly-calculated aspect ratios and calculates the frame width accordingly. The idea behind having a constant height and adjusting the width is that many presentations use rather short lines which typically don't fill the whole width of the page. Changing the width of the frame potentially allows

doi.org/10.47397/tb/44-2/tb137samcarter-beamernews23

the user to switch between aspect ratios without disturbing the layout too much.

4 New `onlytextwidth` class option (v3.65)

The beamer `columns` environment is very convenient to place content side-by-side on a frame. By default, however, the result will most likely have different margins than the surrounding text. Internally, beamer resets the left and right margins to zero within the `columns` environment and then distributes all the remaining space equally before, between, and after the columns. Unless one carefully calculates the column widths to account for this effect, the resulting margins will thus be different from the surrounding text (see Figure 2).

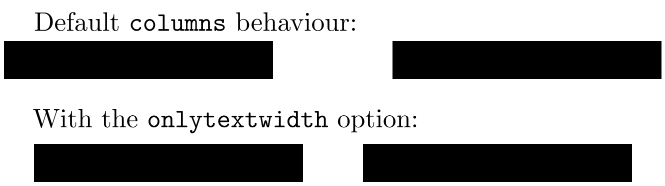


Figure 2: Visualisation of the effect of the `onlytextwidth` option on the `columns` environment.

One can change this locally by using one of the options `onlytextwidth` or `totalwidth=\textwidth` for the `columns` environment. Since beamer v3.65 it is possible to use the new `onlytextwidth` class option to change this behaviour for the whole presentation.

5 The new `s` frame option (v3.65)

Traditionally, beamer offered the `t`, `c` and `b` frame options to influence the vertical position of the frame content.

Erich Schubert contributed the new stretchable frame option `s`. In contrast to the existing options, the `s` frame option does not add any vertical fill at all to the frame. The user has to manually add stretchable material to the frame. This is a bit of extra work, but it allows spreading the content over the whole frame, from top to bottom.

Here's a minimal example of how the new frame option can be used:

```
\begin{frame}[s]
  Text at top
  \vfill
  Text at bottom
\end{frame}
```

6 Modular title page (v3.70)

Many users need to make small adjustments to the title page of their presentation, for example to add the name of their supervisor, the members of a thesis committee, or changing the order in which some of the information is displayed.

Such small changes often resulted in either redefining the whole title page template or tempted the user to resort to dirty hacks.

In version 3.70 of beamer the default title page template is no longer a big monolithic code block, but now uses several smaller templates:

- `title`
- `author`
- `institute`
- `date`
- `titlegraphic`

These templates can be adjusted individually without having to redefine the whole title page.

For example, adding the name of a supervisor below the author name now amounts to simply:

```
\addtobeamertemplate{author}
  {}{Supervisor: Name}
```

7 `page number in head/foot` template (v3.50)

A frequently asked question about beamer is how to change the format of the frame numbers in the foot line. The solution used to be a redefinition of the whole `footline` template.

In the same spirit as making the title page less monolithic, a new `page number in head/foot` template was introduced. Now users can change the appearance of the frame numbers with

```
\setbeamertemplate{page number in head/foot}
  [totalframenumber]
```

The predefined options for this template are

- `default`: the template is empty by default.
- `framenumber`: shows the current frame number.
- `totalframenumber`: shows the current frame number, as well as the total number of frames.
- `appendixframenumber`: similar to the previous, but with separate numbering in the appendix. This option was inspired by the package `appendixnumberbeamer` by Jérôme Lelong.[1]
- `pagenumber`: shows the current page number. The page number can differ from the frame number if overlays are used in the presentation.
- `totalpagenumber`: similar to the `pagenumber` option, but also shows the total number of pages.

[1] `ctan.org/pkg/appendixnumberbeamer`

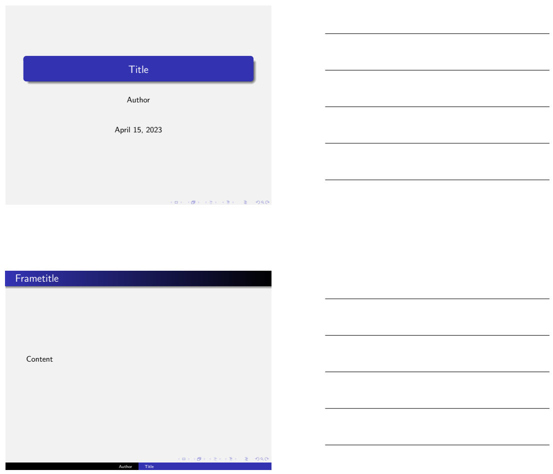


Figure 3: Slides (left) and the corresponding lined note pages (right)

8 Lined `note page` template (v3.64)

Inspired by the `handoutWithNotes` package,[2] beamer now has a `note page` template to add lined note pages to presentations. It can be used via:

```
\setbeamertemplate{note page}[lined]
```

The resulting note pages can be seen in Figure 3.

The user can also adjust the number of lines on the note pages, e.g. with

```
\setbeamertemplate{note page}[lined][5]
```

they will get five lines on the note pages.

9 `lastsection` option for ToC (v3.63)

Beamer already had the `firstsection` option for the table of contents. It allowed having unnumbered sections at the start of the table of contents, which can be useful e.g. for example for an unnumbered introduction.

To allow for unnumbered sections at the end of the table of contents, beamer now also has a `lastsection` option. This option specifies the number of the last numbered section (counted from the first numbered section).

[2] `ctan.org/pkg/handoutwithnotes`

10 New user facing macros (v3.65/70)

Sometimes users might want to know the current aspect ratio of their presentation; for instance, maybe they want to use different background images for the title page depending on the paper format. They can now use the new `\insertaspectratio` macro to access the current aspect ratio.

For users of the `sidebar` theme, two new public macros are available, `\beamersidebarwidth` and `\beamerheadheight`. They provide a way for users to access the value of lengths which were previously only available internally. The new macros are useful if users would like to add logos, etc., to their sidebar and scale them accordingly, or to correct for the asymmetric margins on plain frames with the `sidebar` theme.

11 Calculation of frame geometry (v3.70)

There was also a behind-the-scenes change to how beamer calculates the frame geometry (e.g. the space necessary for the head- and footlines). Previously, the frame geometry was calculated only once, at the start of the presentation. This made it difficult to change between different head- and footlines throughout the presentation. If a user needed a taller headline on their section pages, they had to be very careful to compensate for the additional space or the footline might have been lost.

Now beamer calculates the frame geometry at the start of every frame. This makes the compilation a tiny bit slower, but with much more powerful computers now compared to when beamer was first released in 2003, this change brings new opportunities for creating beamer themes.

12 Summary

The changes presented in this proceeding were cherry picked examples which might be particularly useful for users to know about. Beyond these, many more changes have been made to beamer. A full list of changes is available from the beamer change log[3] or from the CTAN announcements for each new version.[4]

◇ samcarter

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

[3] `github.com/josephwright/beamer/blob/main/CHANGELOG.md`

[4] `ctan.org/ctan-ann/pkg/beamer`

Updating the nostarch class

Boris Veytsman

Abstract

No Starch Press's house style has interesting typographic features. Their implementation posed some TeXnical challenges described in this paper.

1 Introduction

No Starch Press was founded in 1994 by a charismatic publisher, Bill Pollock, who proudly displays on the company's web page a personal story of being hired — and fired — by the major players in the business until he established his own (see `nostarch.com/about`). *No Starch Press* positions its products as *the finest in geek entertainment* and boasts such titles as *Python Crash Course, Python for Kids, How Linux Works,* and *Hacking: The Art of Exploitation*, with topics spanning security, hacking, LEGO, and series like *The Manga Guide*, covering Biochemistry, Calculus, Cryptography, etc., up to the Universe.

It is not surprising that this publisher has strong opinions about the design of their books. Indeed, *No Starch Press* books have a distinct house style, sometimes rather unusual for technical literature. It was an interesting challenge to implement them in LaTeX. I started this task in 2008 with the first release of the `nostarch` class. Since then the publisher's team made many *ad hoc* changes to the original class, suitable for the tasks at hand. Some of these changes assumed manual adjustments of the input, which was error-prone and time consuming. The advent of Overleaf increased the number of authors that used LaTeX for submission, which made these manual adjustments unviable. Thus I was contacted by the publisher with the request to update the class and add new features. In this paper I describe several typographical challenges presented by the class and the ways I approached them.

2 URL breaking

Since the advent of the Internet, URLs have been a nemesis of compositors. A long string of text with no obvious hyphenation points presents an obvious problem for paragraph setting.

The package `url` [1] solves this problem by allowing breaking of URLs. It is a highly customizable package, used internally by the ubiquitous `hyperref` [4], and is the *de facto* standard of URL typesetting in the LaTeX world.

Still, even with this package certain problems remain. For example, how do we split a URL containing a hyphen? If we split `https://hyphenated-` `url.org` before `url.org` (as happened here!), the reader may be confused as to whether the hyphen belongs to the URL or just signifies the break. That is why the `url` package does not allow breaking after hyphens by default: you need to explicitly enable it with the `hyphens` option. *No Starch Press* offers a rather elegant solution: it allows breaks *before* hyphens only, so the reader is not confused by the trailing hyphens.

The full house rules are more complex. Breaks are allowed as follows.

1. *After* the symbols: `# > ] ) } :`
2. *Before* the symbols: `. _ = & - ! ? | , ; @ ' " + < [ ( {`
3. Breaks are not allowed before `/`, so expressions like `https://` are never split.

To implement these rules, we first note that the `url` package typesets URLs as math expressions. The symbols after which breaks are allowed are declared as *mathop* or *mathbin* atoms, and the standard TeX rules for inline math expressions line breaking apply. However, this approach needs a modification if we want to break an expression *before* a symbol. Fortunately, the package provides a `\UrlSpecials` macro, which allows one to associate any behavior with any symbol. Thus we can allow breaks before the dot using the following code:

```
\g@addto@macro{\UrlSpecials}{%
   \do\.{\penalty\UrlBreakPenalty
        \mathchar46\relax}}
```

Note that `\mathchar46` is a dot symbol. Similarly, we can disallow any break before the slash with:

```
\g@addto@macro{\UrlSpecials}{%
   \do\/{\unpenalty\penalty\@M
      \mathchar47\penalty\UrlBreakPenalty}}
```

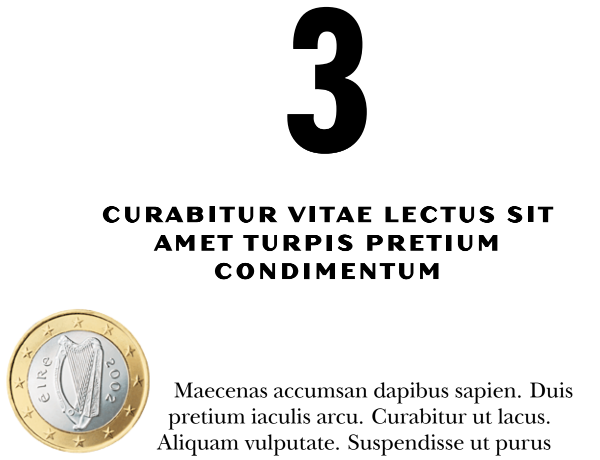
There is an additional problem if the author uses the `amsmath` [2] package. An attempt to redefine the opening bracket leads to the `Bad mathchar (32768)` error message. At TUG'23 David Carlisle explained that the problem is in the macro `\resetMathstrut@` added by `amsmath` to the `\everymath` hook and redefining the character code of the opening bracket. Fortunately, the `url` package has its own hook, `\Url@MathSetup`, called *after* `\everymath`. Thus we can nullify this macro for URLs only:

```
\g@addto@macro{\Url@MathSetup}{%
 \let\resetMathstrut@\relax}
```

3 Chapter opening

The chapters in *No Starch Press* books have quite an impressive opening: the first paragraph is typeset in a larger font, and there is a space for the "circular art", as shown in Figure 1. Happily, LaTeX has

Boris Veytsman
doi.org/10.47397/tb/44-2/tb137veytsman-nostarch



3

CURABITUR VITAE LECTUS SIT AMET TURPIS PRETIUM CONDIMENTUM

Maecenas accumsan dapibus sapien. Duis pretium iaculis arcu. Curabitur ut lacus. Aliquam vulputate. Suspendisse ut purus sed sem tempor rhoncus. Ut quam dui, fringilla at, dictum eget, ultricies quis, quam. Etiam sem est, pharetra non, vulputate in, pretium at, ipsum. Nunc semper sagittis orci. Sed scelerisque suscipit diam. Ut volutpat, dolor at ullamcorper tristique, eros purus mollis quam, sit amet ornare ante nunc et enim.

Phasellus fringilla, metus id feugiat consectetuer, lacus wisi ultrices tellus, quis lobortis nibh lorem quis tortor. Donec egestas ornare nulla. Mauris mi tellus, porta faucibus, dictum vel, nonummy in, est. Aliquam erat volut-

Figure 1: A chapter opening in *No Starch Press* style

a mechanism for setting this, the `\@afterheading` hook in the sectioning macro. The `nostarch` class adds to this hook, among other things, the following code, which uses the switch `\if@firstpara`:

```
\everypar{%
 \if@firstpara
  \rule{\z@}{49\p@}%
  \ifx\@chapterart\@empty
  \else
    \makebox[0pt][r]{%
       \raisebox{-0.5in}[0pt][0pt]{%
       \@chapterart\hspace{0.21in}}}%
   \parshape=5
   0.963in 3.622in   0.913in 3.672in
   0.813in 3.772in   0.563in 4.022in
   0in \textwidth
 \fi
 \fontsize{14pt}{16.5pt}\selectfont\parskip=3pt
 \else
   \parskip=0pt \normalsize\selectfont
  \fi
 \@firstparafalse}
```

Admittedly, this is rather old-fashioned code, using `\everypar`. A better solution would be to use the paragraph hooks relatively recently added to the LaTeX kernel.

4 Captions

The most difficult task so far has been automatic formatting of captions. *No Starch Press* house style does not center figures and tables: they are left justified with figure captions after the figures, and table

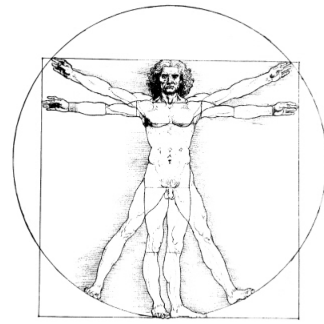


Figure 2-1: Vitruvian man. Note that the caption is typeset in a box with the width automatically calculated from the image.

Maecenas enim. Proin quis neque nec tortor sollicitudin volutpat. Sed at ante. Sed vitae mauris non ante egestas hendrerit. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In venenatis facilisis magna. Phasellus purus. Cras quis mauris. Aliquam eget magna. Donec rutrum sagittis mi. Morbi elementum, est sit amet sollici-

Figure 2: A figure in *No Starch Press* style

captions before the figures. However, a full width caption with a narrow left justified figure body looks rather ugly. Therefore the house style has another requirement: the width of the caption should be no longer than the width of the body, as demonstrated in Figures 2 and 3. Fortunately, the `caption` package [5], used by the `nostarch` class internally, allows typesetting a caption in a parbox of the given width. We just need to calculate this width automatically.

It is relatively easy to do with figures. In most cases figures have just one graphical box. We measure this box and use it to typeset the width of the caption. Note that in TeX the examination of the last box is a destroying operation: we need to return the box to the list if we want to preserve it. We add to the `\endfigure` the following command, `\nostarch@measurecaptionwidth`, defined as:

```
\newcommand\nostarch@measurecaptionwidth{%
 \ifnostarch@overridecaptionwidth\else
 \par
 \setbox\@tempboxa\lastbox
 \setbox\@tempboxa=\hbox{\unhbox\@tempboxa}%
 \global\setlength{%
    \nostarch@captionwidth}{\wd\@tempboxa}%
 \box\@tempboxa\par
 \fi
 \global\nostarch@overridecaptionwidthfalse
}
```

The flag `\ifnostarch@overridecaptionwidth` is discussed below.

The situation with tables is a little bit more complex. When we typeset the caption of a table, we do not know the width of the table, since the body is not yet typeset. Therefore we use the usual LaTeX trick: at the end of a table we save the width of the body to the `.aux` file. At the beginning of the table we check if it has been defined in the previous

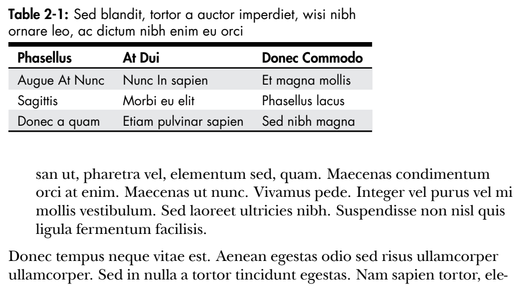


Table 2-1: Sed blandit, tortor a auctor imperdiet, wisi nibh ornare leo, ac dictum nibh enim eu orci

Phasellus	At Dui	Donec Commodo
Augue At Nunc	Nunc In sapien	Et magna mollis
Sagittis	Morbi eu elit	Phasellus lacus
Donec a quam	Etiam pulvinar sapien	Sed nibh magna

san ut, pharetra vel, elementum sed, quam. Maecenas condimentum orci at enim. Maecenas ut nunc. Vivamus pede. Integer vel purus vel mi mollis vestibulum. Sed laoreet ultricies nibh. Suspendisse non nisl quis ligula fermentum facilisis.

Donec tempus neque vitae est. Aenean egestas odio sed risus ullamcorper ullamcorper. Sed in nulla a tortor tincidunt egestas. Nam sapien tortor, ele-

Figure 3: A table in *No Starch Press* style

run, and if yes, we use it. We also need to check whether the current table width is the same as in the previous run. If not, we need to ask the user (or a program like `Makefile` or `llmk` or `arara`) to re-run LaTeX. We save the current widths in macros with the names `\nostarch@tbl@i`, `\nostarch@tbl@ii`, ...: we need to use a separate counter for tables, since `nostarch` numbers the tables per chapter. The following code saves the width of the current table:

```
\nostarch@measurecaptionwidth
\if@filesw
  \immediate\write\@auxout{%
  \gdef\expandafter\noexpand
    \csname nostarch@tbl@\romannumeral
            \c@nostarch@tbl\endcsname
    {\the\nostarch@captionwidth}}%
\fi
\expandafter\ifx
  \csname nostarch@tbl@\romannumeral
          \c@nostarch@tbl\endcsname\relax
  \edef\@tempa{\the\textwidth}%
\else
  \edef\@tempa{\csname nostarch@tbl@
    \romannumeral\c@nostarch@tbl\endcsname}%
\fi
\expandafter\ifdim\@tempa=\nostarch@captionwidth
\else \gdef\nostarch@tbl@warn{%
  \ClassWarningNoLine{nostarch}%
    {Table widths have changed. Rerun LaTeX.}}%
\fi
```

The reading code part performs two tasks. First, it steps the table number `nostarch@tbl`. Second, it sets the width of the caption:

```
\stepcounter{nostarch@tbl}%
\expandafter\ifx
  \csname nostarch@tbl@\romannumeral
          \c@nostarch@tbl\endcsname\relax
  \edef\@tempa{\the\textwidth}%
\else
  \edef\@tempa{%
    \csname nostarch@tbl@\romannumeral
            \c@nostarch@tbl\endcsname}%
\fi
\expandafter\setlength\expandafter
  \nostarch@captionwidth\expandafter{\@tempa}%
```

Another special case is long tables typeset with the `longtable` package [3]. Fortunately, long tables know its width, so we just need to read it. Here is the corresponding code:

```
\ifnostarch@overridecaptionwidth\else
  \global\setlength{\nostarch@captionwidth}{\z@}%
  \bgroup
    \def\LT@entry##1##2{%
      \global\addtolength{%
        \nostarch@captionwidth}{##2}}%
    \csname LT@\romannumeral\c@LT@tables
    \endcsname
  \egroup
\fi
\global\nostarch@overridecaptionwidthfalse
```

Above, we deferred the discussion of the flag `\ifnostarch@overridecaptionwidth`. In fact, no automatic system is 100% accurate. Sometimes a figure or a table contains several boxes of different widths arranged vertically. Sometimes it is too narrow. Thus it makes sense to allow the user to override the algorithm. The command `\NextCaptionWidth` with one argument does just that:

```
\newcommand\NextCaptionWidth[1]{%
  \global\nostarch@overridecaptionwidthtrue
  \global\setlength{\nostarch@captionwidth}{#1}%
}
```

It sets the width of the following caption to the argument and informs the measuring code to skip the measuring.

5 Conclusion

The TeX programming layer, even in the "old" incarnation of LaTeX 2_ε, is quite flexible. It can satisfy many typesetting requirements and provide automatic composition — even when the requirements are rather unusual, as with some of those of *No Starch Press*.

References

[1] D. Arseneau. *The* `url` *package*, 2013. `ctan.org/pkg/url`

[2] LaTeX Project Team. *The* `amsmath` *package*, 2023. `ctan.org/pkg/amslatex`

[3] LaTeX Project Team, D. Carlisle, D. Kastrup. *The* `longtable` *package*, 2021. `ctan.org/pkg/longtable`

[4] LaTeX Project Team, Oberdiek Package Support Group, et al. *The* `hyperref` *package*, 2023. `ctan.org/pkg/hyperref`

[5] A. Sommerfeldt. *The* `caption` *package*, 2023. `ctan.org/pkg/caption`

◇ Boris Veytsman
`borisv (at) lk dot net`
`https://borisv.lk.net`

The LaTeX template generator: How micro-templates reduce template maintenance effort

Oliver Kopp

Abstract

Scientific findings are published by different publishers. These provide different templates. These differ in the documentation and packages provided. For example, `hyperref` or `microtype` are mostly not included or not configured properly. Furthermore, there is a demand for minimal examples in the body of the paper. For instance, how to typeset a listing with line numbers and hyperlink to that line number. These minimal examples should appear in any paper template. If the minimal example is updated, how can various paper templates be updated automatically? The "LaTeX Template Generator" is one answer to this question. It uses "micro-templates" to create full-fledged paper templates containing the same configurations for popular packages. Thus, it reduces the maintenance effort of LaTeX templates.

1 Introduction

In scientific research, starting a new paper often involves using a previous publication as a template. Researchers adapt this base structure to meet their current requirements. However, this practice introduces the potential for inconsistencies, particularly with respect to the usage and configuration of LaTeX packages. Although publishers offer templates for their publication venues, these templates are rather minimal and often omit configuration for `hyperref`, `microtype`, `listings`, etc.

One solution is to offer "extended" templates for each venue, including best practices for each LaTeX package. However, when the package is updated with new features or when new insights about the package emerge, all these templates must be manually updated — a process prone to errors. The "LaTeX Template Generator" (LTG) addresses these challenges. It introduces the concept of "micro templates", each providing a preamble and an example for a specific LaTeX package. These micro templates are then automatically consolidated into templates for various outputs, such as journals, conferences, and student theses. This strategy reduces the risk of errors and simplifies the process of updating LaTeX templates.

Figure 1 presents the roles when working with the LTG:

- The role *package expert* is filled by an individual with extensive knowledge about a specific package. For instance, one expert might specialize in the `hyperref` package, while another might be well-versed in the `microtype` package. These experts contribute not only by configuring the packages, but also by providing minimal examples for each package. This guidance ensures proper usage and assists newcomers in understanding the package functionalities.
- The role *class expert* is held by an individual who understands the necessities of each class. They know which packages are required for each template based on the class and which ones might be unnecessary or counterproductive.
- The role *tooling expert* is held by someone proficient in the respective tooling, such as `latexmk` or `git`. They provide configurations for these tools for templates generated by the LTG.
- The role *content expert* provides guidance on how to write the scientific content of a paper. They may offer advice on structuring arguments, referencing sources, or other aspects of academic writing.
- Finally, the role *template user* describes the ultimate user of the template crafting scientific work.

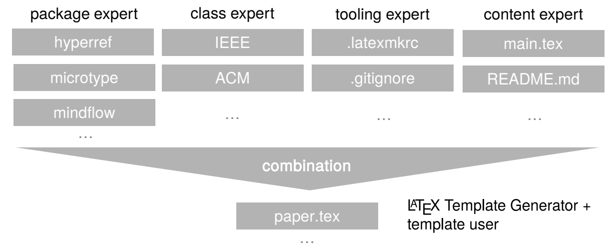


Figure 1: Roles in the process of template generation

The task of combining the different inputs into a template file ("combination" in Figure 1) is done by the LaTeX Template Generator.

The basic idea of the LTG is to offer configuration possibilities where required and to assume sensible defaults where possible. For instance, the template user is offered a choice for the overall template to target (e.g., IEEE or a master's thesis), the language to be used (e.g., English or German), but does not need to choose anything for packages such as `microtype`, because sensible defaults are provided.

In the following, details of the LTG are provided. First, Section 2 presents reasoning on the chosen prompting and generation framework. Section 3 outlines the general concept. Section 4 presents the usage of the LaTeX template generator. Finally, Section 5 provides a discussion and presents an outlook on future work.

doi.org/10.47397/tb/44-2/tb137kopp-microtemplates

2 The choice of yeoman as the basis for the generator

To guide the template user through different options and to generate the final template, an existing framework should be used. The basic requirements are: i) being able to mix multiple micro-templates into larger templates (e.g., the `hyperref` configuration should be stored once in the repository and used by multiple templates) and ii) offering dependent prompts. For instance, if "Overleaf compatibility" is chosen, the choices of the TeX Live variants should be constrained to versions before 2023.

In 2019, the following frameworks were evaluated: Yeoman,[1] Cookiecutter,[2] copier,[3] Jinja2,[4] Cheetah,[5] Apache Velocity,[6] and LuaLaTeX. Some of these options are templating engines only (Jinja2, Cheetah, Apache Velocity). Thus, the prompting would have been required to be hardwired. LuaLaTeX is a very general "framework", which is not commonly used for templating. Since I was not proficient enough to use it, the final choice was between Python-based tooling (Cookiecutter and copier) and JavaScript-based tooling Yeoman. Since both Cookiecutter and copier require the choices to be made available in text-based configuration files and it seemed to be impossible to craft choices dependent on previous choices, I opted for Yeoman.

In 2020, TeXplate was released on CTAN.[7] The current version has a different structure than the LTG. TeXplate relies on TOML[8] files to define the LaTeX file to be generated. LTG builds upon `.tex` files which are "enriched" by templating commands. The claim of LTG is that it is easier for contributors to edit `.tex` files with their editor of choice than to edit TOML files.

All in all, Yeoman has been chosen as the generator framework. It is based on JavaScript and fulfills both requirements: The package preambles and examples are stored in different files (details in Section 3) and offers built-in prompting. Prompt choices can be modified on the fly to enable dependent prompts.

To make the knowledge about the decision sustainable, Markdown Any Decision Records [4] have been written. All the aforementioned options as well as their pros and cons are included as Markdown files inside the path `docs/decisions/`[9] inside the repository.

```
/
└─ generators/
   └─ app/
      ├─ templates/
      │  ├─ ...
      │  ├─ Dockerfile.iot
      │  ├─ Texlivefile
      │  ├─ ...
      │  ├─ main.en.tex
      │  ├─ mindflow.example.en.tex
      │  ├─ mindflow.preamble.en.tex
      │  └─ ...
      ├─ index.js
      └─ options.js
```

Figure 2: Directory tree of the generator

3 The concept of the LTG

This section presents the concept of LTG. Thereby, the file structure of LTG's source repository[10] is used. The most important files are presented in Figure 2.

The pair of files `mindflow.example.en.tex` and `mindflow.preamble.en.tex` show the basic concept of LTG's micro templates using the `mindflow` package. The `mindflow` package[11] is a basic LaTeX package enabling a) quickly noting down thoughts and b) having LaTeX marking these thoughts visually.

A micro template consists of i) a preamble file and ii) optionally an example file. The content of the preamble file is put into the preamble and the example is put as LaTeX example in the document body. The filename is always the package name, followed by either `preamble` or `example` and then `en` for English or `de` for German.

`mindflow.preamble.en.tex` looks as follows:

```
<% switch (documentclass) { case "acmart":
  case "ieee": -%>
\usepackage[incolumn]{mindflow}
<% break; default: -%>
\usepackage{mindflow}
<% break; } -%>
```

On the first line, one sees the templating language "Embedded JavaScript templating" (EJS[12]) in use. In general, template commands are enclosed in

[1] `https://yeoman.io/`
[2] `https://github.com/cookiecutter/cookiecutter`
[3] `https://github.com/copier-org/copier`
[4] `http://jinja.pocoo.org/`
[5] `http://cheetahtemplate.org/`
[6] `http://velocity.apache.org/`
[7] `https://ctan.org/pkg/texplate`
[8] `https://toml.io/en/`
[9] `https://github.com/latextemplates/generator-latex-template/tree/main/docs/decisions`
[10] `https://github.com/latextemplates/generator-latex-template`
[11] `https://ctan.org/pkg/mindflow`
[12] `https://ejs.co/`

`<% ... %>`. The minus sign before the closing `%>` indicates that the following newline should be removed. The intention of the EJS code is that in the case of the document class being a two-column document class (as it is for ACM and IEEE), the `mindflow` package is passed the option `incolumn`. In all other cases, just `\usepackage{mindflow}` is written out.

`mindflow.example.en.tex` looks as follows:

```
<%= heading2 %>{Notes separated from
  the text}

The package mindflow enables writing down
notes and annotations in a way so that they
are separated from the main text.

<%- bexample %>
\begin{mindflow}
This is a small note.
\end{mindflow}
<%- eexample %>
```

The template command `<%= heading2 %>` instructs Yeoman to put the content of the `heading2` variable at that place. In the case of an IEEE template, this is `\subsection`; in the case of, for instance, the provided `scientific-thesis` template, this is `\section`, because the latter's main structuring element is `\chapter`.

The markers `bexample` and `eexample` are markers for LaTeX commands for beginning and ending an example. The LTG defines its own environment for examples to output both rendered LaTeX code as well as the source code. It makes use of the capabilities of the `tcolorbox` package.[13]

The preamble is included as follows in the file `main.en.tex`:

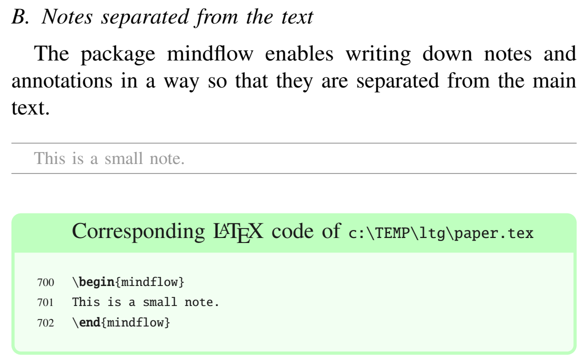
```
<% if (texlive >= 2021) { %><%- include
  ('mindflow.preamble.en.tex', this); } -%>
```

The reason for the guard with the TeX Live version is that `mindflow` was released in 2021, and a template may require support of earlier TeX Live versions.

The file `options.js` contains all options offered to the user. The following excerpt presents the option for TeX Live:

```
{
  type: "list",
  name: "texlive",
  message: "Which TeXLive compatibility?",
  choices(state) {
    const res = [
      {
        name: "TeXLive 2021",
        value: 2021,
      },
      {
        name: "TeXLive 2022",
        value: 2022,
      },
    ];
    if (!state.overleaf) {
      res.push({
        name: "TeXLive 2023",
        value: 2023,
      });
    }
    return res;
  }
},
```

[13] `https://ctan.org/pkg/tcolorbox`



B. Notes separated from the text

The package mindflow enables writing down notes and annotations in a way so that they are separated from the main text.

This is a small note.

Corresponding LaTeX code of `c:\TEMP\ltg\paper.tex`

```
\begin{mindflow}
This is a small note.
\end{mindflow}
```

Figure 3: Mindflow example section in the rendered IEEE template

It starts by defining the option to be a list and instructs that the answer should end up in a JSON object property named `texlive`. The message used for prompting is given in `message`. The choices are created dynamically, based on previous choices. As an example, the user can choose whether the template should be usable on Overleaf. If they opted for "yes", the option to choose TeX Live 2023 is not included, since (at the time of release of the LTG), version 2023 is not supported by Overleaf. (Overleaf typically adds support for new TeX Live releases in the fall [2].) Coming back to the options, there are in total 18 possible options to choose from. Some are dependent on the chosen document class and thus not all are shown to the user.

The file `index.js` calls the prompting, derives internal variables based on the result, and finally creates the resulting files. One internal variable is the final file name. In the case of journals and conferences, this is `paper.tex`. In case of a scientific thesis, it is `main.tex`.

Figure 3 shows the rendered output of this `mindflow` example, when the IEEE template is selected.

4 Usage

The LTG requires a recent Node.js installation. There, the command

```
npm install -g generator-latex-template
```

installs the generator and makes it globally accessible on the target machine. Then, the user can invoke the generator using `yo latex-template`. After issuing that command, LTG outputs following:

```
$ yo latex-template

? Which template should be generated?
    (Use arrow keys)
> Scientic Thesis
  Association for Computing Machinery (ACM)
  Institute of Electrical and Electronics
    Engineers (IEEE)
  Springer's Lecture Notes in Computer
    Science (LNCS)
```

The user is first asked which template they want to create. Currently, a scientific thesis template, ACM, IEEE, and LNCS are supported. More templates are part of future work.

The user navigates through the options using arrow keys. Once a choice is made, the system proceeds to the next question, continuing this iterative process until all questions have been answered.

The following presents the result of an example complete process of selections:

```
? Which template should be generated? IEEE
? Which variant of IEEE paper? conference paper
? Which paper size to use? A4
? Overleaf compatibility? yes
? Which TeXLive compatibility? TeXLive 2022
? Should a Dockerfile be generated?
  yes (Island of TeX)
? Which language should the document be?
  English
? Which package to typeset listings? listings
? Which package to use to "enquote" text?
  csquotes (\enquote{...} command)
? Which package to mark TODOs? pdfcomment
? Include hints on text
  (e.g., how to write an abstract)? Yes
? Include minimal LaTeX examples? Yes
```

After all questions have been answered, Yeoman outputs the files it creates:

```
  create .gitignore
  create .editorconfig
  create paper.bib
  create _latexmkrc
  create localSettings.yaml
  create LICENSE
  create Makefile
  create paper.tex
  create README.md
  create .dockerignore
  create Dockerfile
  create Texlivefile
  create .github\workflows\check.yml
```

Note that the file `latexmkrc` is prefixed by an underscore. This enables uploading the whole repository to Overleaf without any error shown in the user interface. A `Dockerfile` is also generated. In this example, the file is generated based on `Dockerfile.iot` and uses a minimal Island of TeX Docker image [3]. The image installs all LaTeX packages listed in `Texlivefile` into the image. This way, the image size is kept to a minimum.

Finally, a GitHub workflow[14] file is generated. When publishing the repository on GitHub, GitHub's CI will build a docker image based on the `Dockerfile` and build the LaTeX file using `latexmk`.

5 Discussion and outlook

This paper presented the LaTeX Template Generator as one solution to collect knowledge about best practices of packages and a way to include them in rich templates for authors. To add support for a new class, the class expert has to adapt `main.en.tex` and `options.js` to include the class and add proper conditions for packages which should be included or excluded. Then, a new template file is generated. No work for the package experts is caused: Their templates can (most probably) just be used by the class expert. Vice versa, if an update on the package examples are made, the class expert (most probably) does not need to do anything, because the contents are directly available in their template.

For end users, installing Node.js can be tedious. Therefore, for each supported template, a separate GitHub repository is offered. In that repository, default `paper-*.tex` files are offered. For the LNCS template,[15] `paper.tex` uses the Computer Modern font, `microtype` configuration, `listings` configuration (including JSON support), `pdfcomment`[16] for TODO marking, and LaTeX examples. To reduce the size of the `.tex` file, no hints on writing a paper are included.

The repository also offers other `paper-*.tex` files. For instance, `paper-en-times-minted.tex` provides a template where Times New Roman is used for the font and `minted`[17] as the package for listings. A template user can just download the ZIP

[14] https://github.com/features/actions
[15] https://github.com/latextemplates/LNCS
[16] https://ctan.org/pkg/pdfcomment
[17] https://ctan.org/pkg/minted

archive of the repository or even use GitHub's template feature[18] to create a new git repository hosted on GitHub containing the latest template files as single commit.

The most impactful design decision is to have the choices encoded in the templating language. For instance, if LuaLaTeX is chosen by the user, the font configuration is generated for LuaLaTeX. In case the author wants to switch to pdfLaTeX, they must regenerate the whole template: There is no "dynamic" LaTeX if/else construct for a pdfLaTeX fallback. Future work will investigate this further and possibly add an additional user option to generate a more flexible template.

The LTG project itself is a true open source project and calls for contributions of examples of common classes, packages and practices. Currently, around 20 packages and examples are offered. A good start are the hints given by Beeton [1]. As the new *The LaTeX Companion*, third edition, discusses more than 500 examples [5], there is lots of room to include examples. Certainly, a careful selection of discussed packages needs to be made. The LTG focuses on providing only one example per topic. Thus, these examples will surely be enriched by references to TLC3 for the interested readers.

References

[1] B. Beeton. What every (La)TeX newbie should know. *TUGboat* 44(2):164–169, 2023.

[2] T. Hejda. TeX Live and Overleaf revisited. *TUGboat* 44(2):256–256, 2023.

[3] Island of TeX. Living in containers — on TeX live in a docker setting. *TUGboat* 44(2):249–252, 2023.

[4] O. Kopp, A. Armbruster, O. Zimmermann. Markdown architectural decision records: Format and tool support. In *ZEUS*, vol. 2072 of *CEUR Workshop Proceedings*, pp. 55–62, 2018. https://ceur-ws.org/Vol-2072/paper9.pdf

[5] F. Mittelbach, U. Fischer. *The LaTeX Companion: Parts I & II.* Addison-Wesley, third ed., 2023.

◇ Oliver Kopp
Sindelfingen, Germany
https://github.com/koppor
ORCID 0000-0001-6962-4290

[18] https://docs.github.com/en/repositories/creating-and-managing-repositories/creating-a-repository-from-a-template

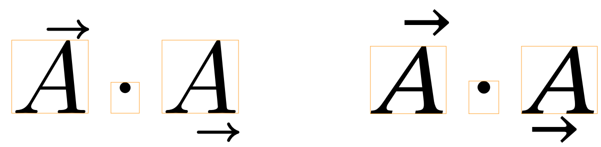
On bottom accents in OpenType math

Hans Hagen, Mikael P. Sundqvist

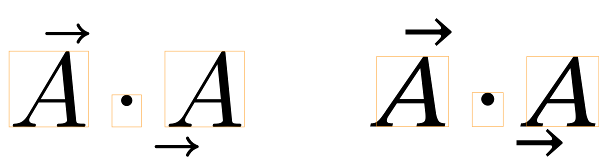
We recently worked on accents in math in ConTeXt. While looking at the bottom accents, we realized that there was work to be done, since Microsoft did not specify how to deal with them (there is support for bottom accents built in to LuaMetaTeX). While discussing and examining examples, we noticed that fonts behave differently, once again.

Let's take the `\wideunderrightarrow` (or simply `\underrightarrow`) as an example. This glyph (`U+20EF`) is missing in the reference font Cambria, but it is available in Latin Modern Math and STIX Two Math, among others. We were surprised to find that the glyph had no width and was positioned with the arrow tip pointing at the x-coordinate zero, but maybe that reflects some previous "standard" on typesetting them. We also saw that in Latin Modern, it had no accent anchor point set, but in STIX Two, it did. Anchor points on the base glyph and the accent are to be aligned.

Until recently, this image shows how it came out in ConTeXt. We show Latin Modern to the left and STIX Two to the right.



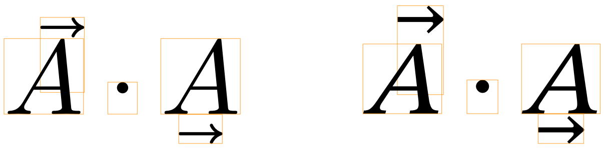
The horizontal location of the top accent is controlled by the anchors of the base character and the accent: they align. We have not changed this behavior because it is, after all, part of the specification. The location of the bottom accent was never specified by Microsoft. Some OpenType fonts mimic old TeX fonts, others mimic Cambria. We wanted a simple model that works well with all fonts. Notice that none of the arrows have orange (grayscaled for print) boxes around them, meaning that they do not carry real widths. The first attempt was to simply mid-align the bottom accents. This came out as follows:



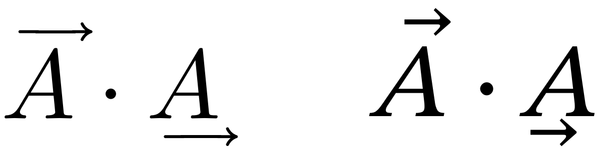
The problem that the arrows do not have a bounding box is now apparent. Looking in FontForge, we found that the tips of the arrows are located at x-coordinate zero. We thus needed to force the arrows

doi.org/10.47397/tb/44-2/tb137hagen-bottomaccent

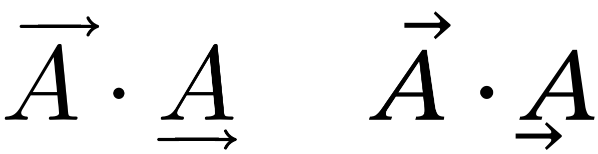
to get their true widths. This was done, and then the centering worked better:



During the process, we wondered if we were making any obvious mistakes. We compared with the LaTeX output of the same examples. With LuaLaTeX we got:



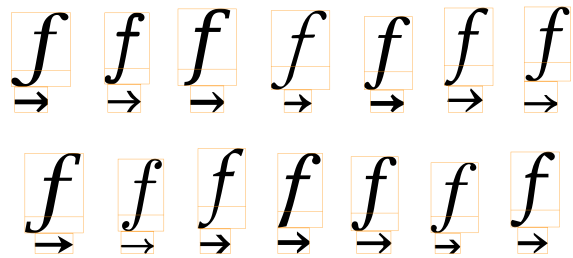
and with XeLaTeX we got:



We notice that Latin Modern (left) worked well in XeLaTeX, but did not look great in LuaLaTeX. On the other hand, STIX Two (right) worked well in LuaLaTeX but not in XeLaTeX. There can be several reasons for this: one can use a traditional TeX engine setup and map OpenType functionality, fonts and parameters to that (maybe that is what XeTeX does) or one can take the traditional fonts, parameters and expectations and translate these to OpenType math rendering (which is what LuaTeX does). Mix that with fonts that are predominantly traditional (Latin Modern) or standard (like Cambria) and you start to see the confusing picture.

For these reasons, the LuaMetaTeX engine adds a lot of detailed control in order to deal with a mismatch. However, the fact we still get unexpected outcomes also points to possible issues (inconsistencies) in fonts. When a designer makes a new math font, a lot of how it behaves depends on what font was taken as its reference.

In the process of getting the best possible output we decided to only use anchor points for the top accents and simply center the bottom anchors under the original box of the glyph. We have discussed elsewhere our getting rid of italic correction (by changing the bounding box and introducing a lower right corner kern). We show here the math italic f in many fonts; it's often one of the more problematic characters, since it sticks out from its box (before we tweak it).



To sum up, there is a problem with how to place bottom accents in Unicode math. The fonts seem to suggest different approaches but the underlying problem lies in the absence of a standardized approach. In light of this, we propose a solution that we hope will effectively address this issue for our users. Depending on the font, an accent has a width or not. When it doesn't have one, we see the mentioned horizontal displacements combined with strange anchors. The displacement sort of positions at the bottom, and the anchor aligns with the character. Because we don't want to rely on side effects we calculate the width from the bounding box and recalculate the anchors. Once more it is more reliable to simply ignore one aspect of OpenType math and individual font implementations.

Our discussion above considers accents below single characters. For multiple characters, we use the variants and extensibles to try to match the total width.

Let us mention one more odd thing that we noticed in connection with this. In STIX Two Math, the bottom accent arrow in fact has five variants and then an extensible recipe. It would have sufficed to provide an extensible recipe and no variants. Moreover, the extensible recipe does not use the base character but the first variant, and that complicates matters if one wants to wipe the variants and go directly to the extensible recipe.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University

A METAFONT for rustic capitals

Victor Sannier

1 Introduction

Littera capitalis rustica (literally *country capital letters*) are a script of the Latin alphabet, the earliest examples of which date back to the first century CE, notably on election posters in the city of Pompeii (see Figure 1). They were gradually regularised in the fourth and fifth centuries and remained in use until the ninth and tenth centuries, but mostly for titles and distinctions [1].

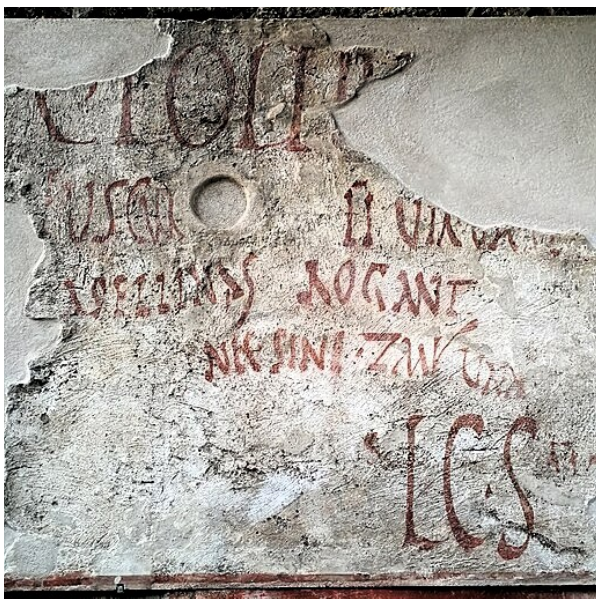


Figure 1: Election poster in Asellina's tavern, Pompeii

Many manuscripts contain splendid examples of rustic capitals. These include the *Vergilius Vaticanus* [7] and the fifth-century *Vergilius Romanus* and *Codex Mediceus*. The first one, an incomplete copy of Virgil's *Aeneid* and *Georgics*, will be the main reference for our work, although we will attempt to rationalise its handwriting with the METAFONT system [3] — while maintaining a sense of authenticity — rather than to reproduce it exactly, imperfection for imperfection. Thus, in the spectrum of type revivals described by Olocco and Patanè [6, Chapter 1], which ranges from literal reproduction to reinvention, we intend to occupy a middle ground.

Whenever possible, we use the same terminology as that found in [2, Chapter 2].

2 Design of the repeating components

Our METAFONT project is divided into several files, `ruscap.mf` defines the macros and characters, while `ruscap10.mf`, `ruscap14.mf`, &c. define the dimensions to be used for rendering to a particular font size. In the rest of this article all values are taken from `ruscap10.mf`.

2.1 Lengths and angles

After numerous measurements on various fragments of the manuscript that we use as a reference, it appeared that the vertical space could be divided into twelve units, from which we also define `s` (for sidebearing) and `o` (for overshoot).

```
u# := 10/12 pt#; % unit
s# := 3/4 u#; % sidebearing
o# := 1/4 u#; % overshoot
```

Then, most capitals occupy nine units above the baseline, some eleven (such as 'F' and 'L'), and the letter 'Q' (and 'G' in one variant) goes down one unit below it. The crossbar in 'E', the junction of the two lobes of 'B', &c. are slightly above half of the upper space.

```
cap_height# := 9 u#;
asc_height# := 11 u#;
desc_height# := 1 u#;
crossbar_height# := 5 u#;
```

Next, the serifs in our font have the shape of a tilde, so all we need to specify is a width and an angle (0 would mean the serif is just a horizontal stroke); see the next section and [3, p. 152].

```
serif_width# := 5/2 u#;
serif_angle := 90 / 6;
```

Two dimensions are also needed for components of some characters, namely the angle of the stroke in the letters 'A', 'N', &c., and the width and angle of the spurs.

```
diag_angle := 90 + 35;
spur_width# := 1/2 u#;
spur_angle := 0;
```

Finally, the virtual pencil used is defined by the thickness of the thickest and thinnest strokes it can draw, and by its slope, here 35 degrees. Rustic capitals are known to be written with a very inclined tool [1].

```
thick# := 5/4 u#;
thin# := 1/3 u#;
pen_angle := 65; % 90 - 35
```

2.2 Macros

Let us give the code of some macros we have written to improve consistency and limit repetition in the project.

doi.org/10.47397/tb/44-2/tb137sannier-rustic

2.2.1 `draw_serif`

To create a serif, we constrain its ends to be horizontally aligned and `serif_width` apart, and draw a line between them with the same initial and final angles.

```
def draw_serif(suffix i, j)(expr width) =
    rt x.j - lft x.i = width;
    y.i = y.j;
    draw z.i{dir serif_angle}
         .. {dir serif_angle}z.j;
enddef;
```

See, for example, the top arm of the letter ‘F’ in Figure 2.

2.2.2 `draw_diag_stroke`

With the spurs defined at both ends of the stroke, all we need to do is set the angle between them and connect all the points. We have increased the tension in the main part so that it is almost straight but still smoothly connects the spurs.

```
def draw_diag_stroke(suffix i, j)(expr a) =
    z.i - z.i.l = z.j.r - z.j
                = spur_width * dir 0;
    z.j - z.i = whatever * (dir angle);
    draw z.i.l .. z.i
         .. tension 3
         .. z.j .. z.j.r;
enddef;
```

In most cases the `angle` parameter will take the value `diag_angle`, but this is not always the case. For example, the two diagonal strokes in the letter ‘M’ don’t have the same angle.

2.2.3 `draw_I`

We use the same code to draw the letters ‘I’ and ‘L’, and the left part of the letters ‘B’, ‘P’, ‘R’, &c.

```
def draw_I(suffix i, j, k, l)(expr sw) =
    x.i = x.j;  % vertical stem
    top y.i = h; bot y.j = 0;
    z.i - z.i.l
      = spur_width * dir spur_angle;

    % Serif
    rt (2 x.j - x.k) - lft x.k
      = serif_width;
    y.j = y.k;
    draw_serif(k, l)(sw);

    draw z.i.l .. z.i .. tension 5 .. z.j;
enddef;
```

The way we constrain the position of the serif ends may require some explanation. The `sw` variable stores the total width of the serif to be drawn. Generally it is not centred around the stem; the right part can be as long as needed and only the width of the left part is constant and should be half the value s of `serif_width`. To achieve this, the following equation should hold:

$$x_k + 2(x_j - x_k) = x_k + s$$

which, after rewriting and taking into account the size of the pencil nib, gives the above code.

3 Design of the characters and kerning

3.1 Example of the letter ‘T’

```
beginchar("T", 6u# + 2s#, cap_height#, 0);
    "Rustic␣T";
    pickup rustic_pen;
    x1 = w - x2; top y1 = h;
    draw_serif(1, 2)(w - 2s);
    x3 = w - x4; bot y3 = 0;
    draw_serif(3, 4)(serif_width);
    draw 1/2 [z1, z2] .. 1/2 [z3, z4];
    labels(range 1 thru 4);
endchar;
```

3.2 Example of the letter ‘N’

The letters ‘M’ and ‘N’ are the most involved we have designed, but the code is still straightforward.

```
beginchar("N", 7u# + 2s#, cap_height#, 0);
    "Rustic␣N";
    pickup rustic_pen;

    % Diagonal stroke
    x1 + x2 = w; top y1 = h; bot y2 = 0;
    draw_diag_stroke(1, 2)(diag_angle);

    % Left stem
    bot y3 = 0; lft x3 = s;
    draw z1 .. z3;
    1/2 (z3.l + z3.r) = z3;
    draw_serif(3.l, 3.r, serif_width);

    % Right stem
    x4 = x5 = x2.r;  % vertical stem
    top y4 = h; bot y5 = 0;
    z4 - z4.l
      = spur_width * dir spur_angle;
    draw z4.l .. z4 -- z5;
```

```
    labels(range 1 thru 5);
endchar;
```

Perhaps the most debatable choice we have made is the angled left-hand stem and straight right-hand stem, but we think it works well with letters such as ‘A’ and ‘L’.

3.3 Example of the letter ‘S’

The design of the letter ‘S’ is peculiar in that it consists of a single stroke and doesn’t call any of our custom macros.

```
beginchar("S", 4u# + s#, cap_height#, 0);
    "Rustic␣S";
    pickup rustic_pen;
    lft x2 = s; x2 = x4;
    rt x1 = w; x1 = x3;
    top y1 = h - u; bot y4 = o;
    h - y2 = y3;
    z2 - z3 = whatever * dir diag_angle;
    draw z1{curl 2} .. z2
         .. z3 .. {curl 1}z4;
endchar;
```

Note the slight asymmetry introduced by a different curl value at each end.

3.4 Side-bearing and kerning

We chose a uniform side-bearing throughout the typeface and corrected glaring kerning problems with a ligature table.

```
ligtable "A": "C" kern -.5u#, "T" kern -u#;
ligtable "K": "O" kern -.5u#;
ligtable "L": "O" kern -.5u#, "T" kern -u#;
ligtable "N": "V" kern .5u#;
...
```

We believe that fine-tuning the interletter spacing does not align with our goal of authenticity.

3.5 Remarks

While scanning various manuscripts and online resources, we came across two variants of the letter ‘G’ [5, 8]. In the final design, we chose the one that looks more like a square capital as the main variant, and made the other available as ‘g’.

4 Comparison with other typefaces

See Figures 3, 4 and 5 for a comparison of fragments of *Vergilius Vaticanus* with the typefaces designed by (a) Landers [4], (b) Wilson [9] and (c) the author respectively. The (b) and (c) typefaces were created with the METAFONT system, (a) was not.

While the former designs may be well suited to their authors’ aims, they do not correspond to the one we stated in the introduction. In particular, we would like to draw the reader’s attention to the following points:

- in (a), the letters are slightly angled to the right,
- in (a), the letters ‘E’ and ‘N’ are not the same height, and neither are the letters ‘S’ and ‘U’,
- in (a), the letters ‘D’ and ‘R’ feature a large spur,
- in (b), the crossbar of the letter ‘E’ is long and sinuous,
- in (b), the strokes become significantly thicker as they descend,
- in (a) and (b), some paths are quite steep, such as the letter ‘D’ in (a) and the letter ‘G’ in (b).

Figure 2: Character set of the `ruscap` typeface

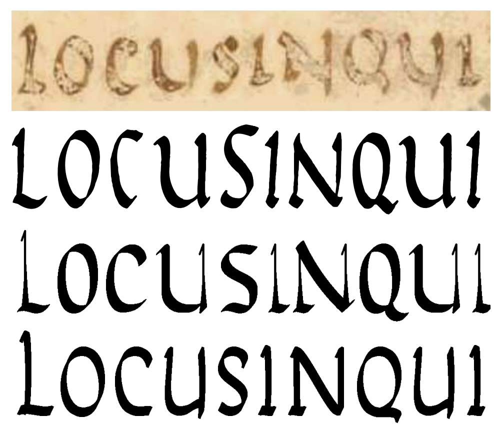


Figure 3: The words “locus in qui” rendered in different typefaces

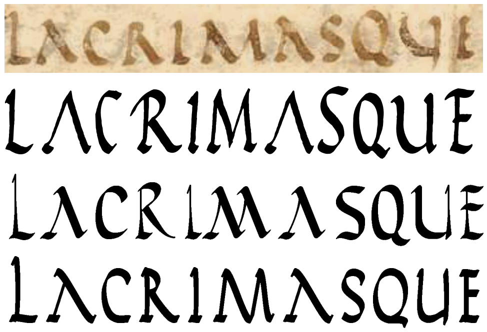


Figure 4: The word "lacrimasque" rendered in different typefaces

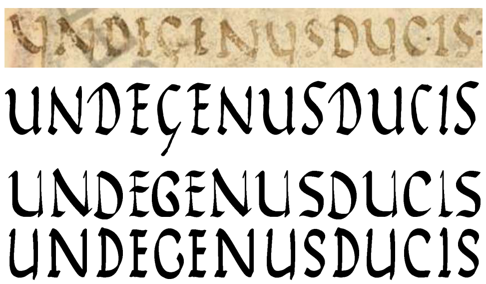


Figure 5: The words "unde genus ducis" rendered in different typefaces

It also seems to us that our METAFONT code is simpler than that written by Wilson, partly, but not only, because our character set is smaller.

5 Conclusion and future work

In this article, we have limited ourselves to presenting just one font from the ruscap family, but METAFONT makes it easy to achieve a variety of designs, including different weights, by changing just a few parameters. For example, the result of setting a uniform pen thickness, a serif angle of 0 and a slightly higher crossbar height is shown in Figure 6.

In the near future, we plan to iterate on the shapes of some characters (particularly 'M' and 'U'), consider the revisions made during the TUG 2023 event, and finally submit our typeface to the Comprehensive TeX Archive Network (CTAN) for everyone to use freely.

We will also consider extending the character set to include the two "Ramist letters" 'J' and 'U', the letter 'W' and the Indo-Arabic digits.

Figure 6: The word 'ciceronianus' rendered in two fonts of the `ruscap` family

Acknowledgement

I would like to thank the GUTenberg Association for funding my participation in the 44th annual meeting of the TeX Users Group, and its president Patrick Bideault, who provided me with Olocco and Patanè's excellent little book.

References

[1] Bibliothèque Nationale de France. La rustica, 2003. Educational dossier accompanying the exhibition "Jean Fouquet : peintre et enlumineur du XVe siècle". `http://expositions.bnf.fr/fouquet/reperes/32/ecriture/rustica.htm`

[2] K. Cheng. *Designing Type*. Yale University Press, second ed., [2005] 2020.

[3] D.E. Knuth. *The METAFONTbook*. American Mathematical Society/Addison-Wesley Publishing Company, [1986] 2021.

[4] J. Landers. Rustic capitals, 2000. MouserFonts.

[5] J.J. Marcos. Manual of Latin Paleography, 2017. `http://guindo.pntic.mec.es/jmag0042/LATIN_PALEOGRAPHY.pdf`

[6] R. Olocco, M. Patanè. *Designing Type Revivals*. Lazy Dog Press, 2022.

[7] P. Vergilius Maro. Opera fragmenta, [19 BCE] c. 400 CE. Late antique illuminated manuscript. `digi.vatlib.it/view/MSS_Vat.lat.3225`

[8] S. Vittori. De scriptione capitali rustica, 2020. `youtube.com/watch?v=xcGHhBQXLNQ`

[9] P.R. Wilson. Roman rustic manuscript book-hand font, [1999] 2001. `ctan.org/pkg/rustic`

◇ Victor Sannier
GUTenberg Association

An updated survey of OpenType math fonts

Ulrik Vieth

Abstract

OpenType math fonts were introduced more than 15 years ago. Over the years, more and more math fonts have been developed and added to the font collection. In this paper, we review some of the more recent additions, comparing them to previous choices of OpenType math fonts such as Cambria, Lucida, Latin Modern, and TeX Gyre.

In our analysis, we focus on completeness of math symbols and alphabets, and on design choices of math alphabets. A detailed technical study of glyph and font metrics is beyond the scope of this paper, but some aspects of this have been recently addressed by other contributions.

1 Introduction

OpenType math fonts were introduced more than 15 years ago. It started when Microsoft added support for math typesetting in Office 2007 [1] and proposed an extension of the OpenType font format, adding a MATH table. This eventually became part of the OpenType standard [2].

It did not take long before the TeX community recognized the potential of OpenType math fonts [3, 4] and started adopting the font technology for their own purposes.

XeTeX started in 2008 to introduce limited support for OpenType math in the scope of an extended TeX math engine [5]. LuaTeX followed in 2009 with a more complete implementation, aiming to provide a full-featured OpenType math engine [6, 7].

Since 2010 both engines and supporting macro packages and font loaders have been available in the mainstream TeX Live distribution. At this point, the technology for OpenType math typesetting was essentially ready for use, except that there weren't many OpenType math fonts available yet.

2 Overview of available math fonts

When OpenType math was introduced, only a single math font was available: Cambria Math [8] by Tiro Typeworks, commissioned by Microsoft and distributed as a system font with Office 2007. Cambria Math was intended as a reference implementation showcasing the features of OpenType math, illustrated in a promotional booklet.

This was followed in 2008 by Asana Math [9] by Apostolos Syropoulos as the first independently developed OpenType math font, which was based on `pxfonts` by Young Ryu.

When the STIX fonts 1.0 were released in 2010, they were quickly assembled as an OpenType math font and released as the XITS fonts [10]. It was only years later that OpenType versions of STIX fonts were released with STIX fonts 1.1.1 in 2013 and the much revised STIX2 fonts in 2016 [11, 12]. At this time, the earlier XITS and STIX fonts are considered obsolete; only the STIX2 fonts are maintained.

Perhaps the most significant contribution to the collection of math fonts came in 2011–2014 with the development of the Latin Modern and TeX Gyre math fonts by the GUST font team with support from various TeX user groups [13, 14, 15, 16].

Another contribution by the GUST team was the development of a math font for DejaVu in 2015, which was added to the TeX Gyre collection.

During this same time also came the development of Lucida OpenType text and math fonts in 2011–2012, initiated as a TUG project with support from Bigelow & Holmes and a group of volunteers. While the Lucida fonts are not free (in either sense), they are available at a very reasonable price from TUG [17].

With these developments, there were already more than 10 choices of OpenType math fonts in 2015, while there had been just a few in 2010.

But there was more to come: In the following years, more math fonts were added, complementing various freely available OpenType text fonts.

Starting in 2016, Khaled Hosny developed the Libertinus OTF fonts [18], derived from Libertine and Biolinum, and added a Libertinus Math font, borrowing some symbols and alphabets from other existing fonts such as the STIX fonts.

A Garamond Math font [19] followed in 2018, developed by Yuansheng Zhao, using alphabets from EB Garamond and borrowing a sans-serif alphabet from Libertinus Math.

Daniel Flipo provided the Erewhon Math and XCharter Math fonts [20, 21], using alphabets from Michael Sharpe's Erewhon[1] and XCharter text fonts, which, in turn, are derived from extended versions of Adobe Utopia and Bitstream Charter. The math symbols for Utopia and Charter are based on the Fourier-GUT and MathDesign packages by Michel Bovani and Paul Pichaureau.

Another recent contribution by Daniel Flipo is the KpFonts OTF collection [22], based on KpFonts by Christophe Caignaert, which, in turn, is derived from a version of URW Palladio (not Kepler!) and complemented by a sans-serif and a monospace to make a complete font family.

[1] *erewhon* backwards is *nowhere*, which alludes to Utopia.

doi.org/10.47397/tb/44-2/tb137vieth-otmath

font name	first	latest	version	release	sources	developer, maintainer	ref.
Cambria Math	2007	2019	6.99	MS	—	Microsoft, Tiro Typeworks	[8]
Asana Math	2008	2019	0.958	CTAN	—	Apostolos Syropoulos	[9]
XITS Math	2010	2020	1.302	CTAN	Github	Khaled Hosny	[10]
STIX Math (obsolete)	2010	2014	1.1.1	CTAN	Github	David Jones, STIpub	[11]
STIX Two Math	2016	2021	2.13	CTAN	Github	David Jones, STIpub	[12]
Latin Modern Math	2011	2014	1.959	CTAN	GUST	GUST font team	[13]
TeX Gyre Pagella Math	2012	2016	1.632	CTAN	GUST	GUST font team	[14]
TeX Gyre Termes Math	2012	2016	1.543	CTAN	GUST	GUST font team	[14]
TeX Gyre Bonum Math	2013	2016	1.005	CTAN	GUST	GUST font team	[14]
TeX Gyre Schola Math	2014	2016	1.533	CTAN	GUST	GUST font team	[14]
TeX Gyre DejaVu Math	2015	2016	1.106	CTAN	GUST	GUST font team	[14]
Lucida Bright Math	2012	2023	1.901	TUG	—	Bigelow & Holmes, TUG	[17]
Libertinus Math	2016	2021	7.040	CTAN	Github	Khaled Hosny	[18]
Garamond Math	2018	2022	2022-01	CTAN	Github	Yuansheng Zhao	[19]
Erewhon Math	2019	2023	0.63	CTAN	—	Daniel Flipo	[20]
XCharter Math	2022	2023	0.50	CTAN	—	Daniel Flipo	[21]
KpFonts (Roman, Sans)	2020	2023	0.55	CTAN	—	Daniel Flipo	[22]
GFS Neohellenic Math	2016	2022	1.02	CTAN	—	Antonis Tsolomitis, GFS	[23]
Fira Math	2018	2020	0.3.4	CTAN	Github	Xiangdong Zeng	[24]
Lato Math	2020	2020	0.1	—	Github	Chenjing Bu	[25]
Noto Math	2020	2023	2.539	—	Github	Noto Fonts Project	[27]
New CM Math	2019	2023	4.6	CTAN	—	Antonis Tsolomitis	[29]
Concrete Math	2022	2023	0.50	CTAN	—	Daniel Flipo	[30]
Euler Math	2022	2023	0.50	CTAN	—	Daniel Flipo, Khaled Hosny	[31]

Table 1: List of available OpenType math font packages with dates of first and latest releases, latest versions, availability of releases and sources, developer or maintainer, as well as links to resources.

While KpFonts also includes a sans-serif design, it is not the only sans-serif math font available.

There is GFS Neohellenic Math [23], maintained by Antonis Tsolomitis, which is based on a sans-serif font in neo-hellenic style that was developed by the Greek Font Society (GFS).

Another example is Fira Math [24] developed by Xiangdong Zeng in 2018, using alphabets from Fira Sans and corresponding math symbols.

There also exists a project for Lato Math [25], using alphabets from Lato [26] by Łukasz Dziedzic combined with symbols borrowed from Fira Math. Unfortunately, the project seems unfinished and is unsuitable for distribution in the current state.

Another very recent project, started in 2023, aims to provide OpenType math functionality for Noto Math [27]. While the font already exists for some years, it only provided the glyphs, but it didn't come with a MATH table, so was lacking usable math typesetting functionality. When the project is done, it will provide another important addition to the collection of sans-serif math fonts.

Finally, besides all the developments to provide math support for various existing OpenType fonts, there has also been renewed interest in extending and reviving some traditional TeX fonts.

A significant extension is the New Computer Modern font family [28, 29] by Antonis Tsolomitis, which extends Latin Modern fonts in many ways. Besides numerous additions to the text fonts, it also adds additional Unicode blocks of mathematical and technical symbols to the math fonts. As a result, these fonts are now the most complete math fonts, even more complete than STIX fonts.

Another recent contribution by Daniel Flipo has revived some traditional TeX fonts, providing OTF versions of Concrete Math and Euler Math [30, 31]. While Concrete Math was generated from sources, Euler Math is based on Neo Euler [32] by Khaled Hosny, started in 2009, which originated from a collaboration with Hermann Zapf more than a decade ago but was long since abandoned [33].

With these developments, we now have more than 20 choices of OpenType math fonts in 2023 (not counting variants). This is a significant increase compared to the numbers of 2015 or 2010.

A summary of available OpenType math font packages is provided in table 1.

font name	weights
XITS Math	Regular, Bold
Lucida Bright Math	Regular, Demi
Erewhon Math	Regular, Bold (minimal)
XCharter Math	Regular, Bold (minimal)
KpRoman Math	Light, Semibold
	Regular, Bold
KpSans Math	Regular, Bold
New CM Math	Regular, Book

Table 2: List of available OpenType math fonts which provide bold versions or additional weights.

Some OpenType math font packages come with multiple weights, so the total number of individual font shapes is now more than 30.

In some cases, there is a fairly complete bold math font, in other cases, only a bare minimum is provided, suitable for inline math only.

Besides bold math fonts, there are also some font packages which provide multiple weights of the base fonts, such as light or book variants. A summary of OpenType math fonts with bold or additional weights is provided in table 2.

Nearly all OpenType math fonts discussed in this paper are free and readily available from CTAN or TEX Live. However, some unfinished projects are currently only available from Github.

The only non-free fonts discussed in this paper are Cambria Math, which comes as a system font on Windows, and Lucida, which are sold via TUG. We have excluded other non-free fonts since we don't have any up-to-date information.

In this paper, we want to analyze how the available math fonts compare with regard to coverage of symbols and alphabets, and with regard to design choices of alphabets.

Some of these topics have also been considered in an earlier review [34], which reflected the state of math fonts in 2012, when just a few OpenType math fonts were available, such as Cambria, Lucida, Latin Modern, and some TEX Gyre fonts.

In this review, we provide an update on the state of OpenType math fonts in 2023 with many updated and many additional fonts available. Given the number of available fonts, a detailed technical study of font parameters and glyph metrics is beyond the scope of this paper.

Fortunately, some recent studies by the LuaMetaTEX developers[2] have covered this topic in detail and have also resulted in improvements or repairs of several OpenType math fonts [35, 36, 37].

[2] LuaMetaTEX (LMTX) is a follow-up of LuaTEX.

3 Completeness of available math fonts

In the following sections, we want to analyze how the available math fonts compare with regard to completeness of symbols and alphabets. In order to determine the range of coverage, we are essentially counting the number of Unicode slots provided in a given OpenType font.

This could be done using a test script such as Frank Mittelbach's `unicodefonttable` package [38, 39], which generates a Unicode font table for a given font and counts the available glyphs.

A similar approach, more specific to math fonts, would be to adapt the `unimath-symbols.ltx` table from the documentation of the `unicode-math` package [40], which typesets a font table of Unicode math symbols encoded in `unicode-math-table.tex` and counts the available glyphs.

For our project, we used a modified version of the latter, providing separate counts for symbols and alphabetic characters. We have also used a modified version of the symbol table.

The numbers determined this way represent a lower estimate for the available glyphs, since we are only counting the base glyphs in Unicode slots and only the known symbols.

In most cases, OpenType math fonts provide more than the base glyphs. For big operators, big delimiters, wide accents, and similar objects, there are multiple sizes and extensible versions.

Besides additional sizes, many OpenType math fonts also provide additional glyph variants that can be accessed via stylistic sets.

It is difficult to determine an exact number of glyphs that should be provided to make a math font complete. The boundary between mathematical and technical symbols is a little vague and the decision of which symbols to include or exclude in the encoding table is somewhat subjective.

Furthermore, Unicode makes new releases every year, so there could be additional symbols added from time to time, which could be overlooked if they are missing in the symbol table.

Some of the most complete OpenType math fonts amount to 1270 symbols and 1170 alphabetic characters, so there would be 2440 glyphs in total, not counting any sizes or variants. If we include the additional sizes and variants, there will be even more glyphs needed for a complete math font.

While the glyph variants are usually hidden and excluded from the count, some font designers also make them available in the private-use area, which would add them to the count of Unicode slots, and the total count could be misleading.

3.1 Completeness of math symbols

When analyzing the counts regarding completeness of math symbols, we find that there are essentially two groups of OpenType math fonts.

The first group aims for completeness, covering more or less the complete range of Unicode math, providing some 1150–1270 math symbols:

New CM Math	1270 symbols
STIX Two Math	1256 symbols
XITS Math	1253 symbols
Lato Math	1221 symbols
Asana Math	1211 symbols
GFS Neohellenic Math	1175 symbols
Noto Math	1162 symbols
Cambria Math	1157 symbols
Lucida Bright Math	951 symbols

In this group we find fonts that were designed for completeness such as STIX/XITS, Noto, or Lato, but also some new entries such as New CM Math, which is currently the most complete math font. Cambria is also fairly complete by now, after being much less complete in earlier versions. Lucida is somewhere in between: It is a little behind the first group, but way ahead of the second group.

The second group does not aim for completeness and covers only a subset of symbols, providing some 500–600 math symbols:

Erewhon Math	607 symbols
Garamond Math	604 symbols
Euler Math	602 symbols
Concrete Math	600 symbols
XCharter Math	600 symbols
KpFonts (Roman, Sans)	589 symbols
Libertinus Math	560 symbols
TeX Gyre Math (5×)	556 symbols
Latin Modern Math	554 symbols
Fira Math	508 symbols

Among this group, the Latin Modern and TeX Gyre math fonts by the GUST font team provide a consistent subset across all fonts, which could be taken as a starting point for a common subset encoding. Unfortunately, there is not much agreement among other fonts, so the details of symbol coverage will be slightly different for each font.

While a subset of 500–600 math symbols may seem small compared to the full Unicode symbol range, it is not that small in practice. If we consider that a traditional TeX with AMS fonts had no more than 5 fonts of 128 slots to encode all the math symbols and alphabets, any OpenType font with 500–600 symbols (not including alphabets) will be as good as any traditional TeX font.

Finally, it is interesting to note how bold math fonts compare, if they are provided at all.

Since the regular math fonts already include bold alphabets for markup, separate bold fonts are only needed in the context of headings, when formulas are switched to bold as a whole.

As shown in table 2, only a few font packages provide a separate bold math font, and these bold versions come with a smaller range of math symbols compared to the regular versions:

XITS Math Bold	499 symbols
KpFonts (Roman, Sans)	495 symbols
Lucida Bright Math Demi	478 symbols
Erewhon Math Bold	124 symbols
XCharter Math Bold	116 symbols

In the cases of Erewhon Math and XCharter Math, the idea of only providing support for inline math was taken to the extreme, omitting most of the big operators and big delimiters, and only including the basic sizes of the most common symbols.

3.2 Completeness of math alphabets

When analyzing the counts regarding completeness of math alphabets, we find that there are again several groups of OpenType math fonts.

The first group aims for completeness, covering all of the math alphabets, providing some 1150–1170 alphabetic symbols:

New CM Math	1170 alphabetic
STIX Two Math	1170 alphabetic
XITS Math	1170 alphabetic
Cambria Math	1170 alphabetic
Asana Math	1167 alphabetic
Noto Math	1164 alphabetic
TeX Gyre Math (5×)	1163 alphabetic

The second group is a little less complete, covering most of the math alphabets with some limitations, providing some 1050–1150 alphabetic symbols:

Libertinus Math	1145 alphabetic
Erewhon Math	1117 alphabetic
Latin Modern Math	1111 alphabetic
XCharter Math	1108 alphabetic
Concrete Math	1107 alphabetic
Garamond Math	1100 alphabetic
Euler Math	1088 alphabetic
KpFonts (Roman, Sans)	1070 alphabetic
Lucida Bright Math	1038 alphabetic

Among the most common omissions are lowercase Script and BBold, which are missing in several fonts. Lucida Math is missing only lowercase bold Script and bold Fraktur. Garamond Math is missing lowercase Greek in sans serif bold italic.

font name	regular up it bf bi	sans-serif up it bf bi	Script scr bscr	Calligr. cal bcal	Fraktur frak bfrak	BBold bb	Mono tt
Cambria Math	• • • •	• • • •	• • • •	– – – –	• • • •	• •	• •
Asana Math	• • • •	• • • •	• • • •	– – – –	• • • •	• •	• •
XITS Math	• • • •	• • • •	• • • •	• – • –	• • • •	• •	• •
STIX Two Math	• • • •	• • • •	• • • •	• – • –	• • • •	• •	• •
Latin Modern Math	• • • •	• • • •	• – • –	– – – –	• • • •	• •	• •
TEX Gyre Math (5×)	• • • •	• • • •	• • • •	– – – –	• • • •	• •	• •
Lucida Bright Math	• • • •	• • • •	• • • –	• – • –	• • – –	• –	• •
Libertinus Math	• • • •	• • • •	• • • •	– – – –	• • • •	• •	• •
Garamond Math	• • • •	• • • ◦	• • • •	• – • –	• • • •	• •	• •
Erewhon Math	• • • •	• • • •	• – • –	– – – –	• • • •	• •	• •
XCharter Math	• • • •	• • • •	• – • –	– – – –	• • • •	• •	• •
KpRoman Math	• • • •	• • • •	• – • –	• – • –	• • • •	• –	• •
KpSans Math	• • • •	• • • •	• – • –	• – • –	• • • •	• –	• •
GFS Neohellenic Math	• • • •	– – – –	• – – –	– – – –	• – – –	• –	– –
Fira Math	• • • •	– – – –	– – – –	– – – –	– – – –	• •	• •
Lato Math	• • • •	– – – –	– – – –	– – – –	– – – –	• •	• •
Noto Math	• • • •	• • • •	• • • •	– – – –	• • • •	• •	• •
New CM Math	• • • •	• • • •	• • • •	– – – –	• • • •	• •	• •
Concrete Math	• • • •	• • • •	• – • –	– – – –	• • • •	• •	• •
Euler Math	• • • •	• • • •	• – • –	– – – –	• • • •	• •	• •

Table 3: List of available OpenType math fonts with coverage of math alphabets.
For regular and sans-serif the columns indicate upright, italic, bold and bold italic.
For Script, Calligraphic, Fraktur, BBold, the columns indicate upper- and lowercase.

In previous versions, Concrete and Euler lacked the sans-serif and typewriter alphabets, but these have recently been added.

Since Euler is an upright design, it used to have a special setup, which only provided the upright and bold slots, while it left out the italic and bold italic slots. In the most recent version, the italic slots are now substituted with the upright symbols.

The third group consists of sans-serif designs, which leave out the sans-serif slots, resulting in much lower numbers:

Lato Math	606 alphabetic
Fira Math	584 alphabetic
GFS Neohellenic Math	568 alphabetic

Again, the most common omissions are lowercase Script and BBold, which are missing in several fonts. GFS Neohellenic is missing lowercase and bold Script and Fraktur, as well as lowercase BBold. Lato and Fira are missing all of Script and Fraktur, but they do provide a full set of BBold.

While many sans-serif fonts provide a reduced set of math alphabets, Noto is an exception: it provides the complete range of math alphabets, but uses an inconsistent setup. While the upright uses a sans-serif, the italic, bold, and bold italic use a serif variant.

Finally, it is interesting to note how bold math fonts compare in terms of math alphabets.

When formulas are switched to bold as a whole in the context of headings, regular alphabets should be replaced by bold alphabets, and bold alphabets should ideally become ultrabold, if available, but in most cases they just remain bold.

The numbers of alphabetic symbols for bold fonts, which sometimes leave out typewriter slots, are usually a little lower than for regular fonts:

XITS Math Bold	1093 alphabetic
KpFonts (Roman, Sans)	1066 alphabetic
Erewhon Math Bold	1001 alphabetic
XCharter Math Bold	1001 alphabetic
Lucida Bright Math Demi	961 alphabetic

Gaps in the regular fonts are usually reflected in the bold fonts: Lucida is already missing lowercase bold Script and bold Fraktur in the regular font, so the bold font is also missing Script and Fraktur.

Finally, some bold fonts which only provided a minimal set used to leave out the bold alphabets when the regular alphabets were switched to bold, but this practice has now been discontinued.

A summary of available or missing alphabets in the various math fonts and bold math fonts is provided in tables 3 and 4.

font name	regular up it bf bi	sans-serif up it bf bi	Script scr bscr	Calligr. cal bcal	Fraktur frak bfrak	BBold bb	Mono tt
XITS Math Bold	• • • •	• • • •	• • • •	• – • –	• • • •	• •	– –
Lucida Bright Math Demi	• • • •	• • • •	• – • –	• – • –	– – – –	• –	• •
Erewhon Math Bold	• • • •	• • • •	• – • –	– – – –	• • • •	• –	– –
XCharter Math Bold	• • • •	• • • •	• – • –	– – – –	• • • •	• –	– –
KpRoman Math Bold	• • • •	• • • •	• – • –	• – – –	• • • •	• –	• •
KpSans Math Bold	• • • •	• • • •	• – • –	• – – –	• • • •	• –	• •

Table 4: List of available OpenType bold fonts with coverage of math alphabets.
For regular and sans-serif the columns indicate upright, italic, bold and bold italic.
For Script, Calligraphic, Fraktur, BBold, the columns indicate upper- and lowercase.

While it may be difficult to keep track of the details, users of OpenType math fonts shouldn't be too concerned about missing alphabets, unless they have special requirements.

In general, OpenType math fonts provide more math alphabets than traditional TEX math fonts, and most of the gaps only affect specific alphabets, which may not be used much.

It should be safe to assume that nearly all OpenType math fonts provide at least the main alphabet in 4 shapes, including Latin and Greek, as well as a basic set of Script, Fraktur, and BBold.

There may be gaps when it comes to lowercase Script, lowercase BBold, bold Script or bold Fraktur, but these are much less used. There may also be gaps in the sans-serif or typewriter alphabets.

4 Design choices of math alphabets

For a full-featured OpenType math font, a number of math alphabets are required:

- 4 shapes of the main font (upright, italic, bold, bold italic), each including Latin and Greek,
- 4 shapes of a sans-serif (upright, italic, bold, bold italic), some including Latin and Greek,
- 2 shapes of Script/Calligraphic (regular, bold), each including upper- and lowercase,
- 2 shapes of Fraktur/Blackletter (regular, bold), each including upper- and lowercase,
- 1 shape of Blackboard bold or BBold (regular), also including upper- and lowercase,
- 1 shape of a monospace/typewriter (regular), also including upper- and lowercase.

To provide all these alphabets, it will be necessary to assemble glyphs from multiple sources and to adjust them to match the main font.

When dealing with a comprehensive font family, some choices may be obvious, such as choosing a sans-serif or a typewriter font, but in most cases some design decisions will be needed.

In the following sections, we want to consider how the available OpenType math fonts compare with regard to design choices of math alphabets for Script, Fraktur, and Blackboard Bold.

While some design choices in existing fonts may be unfortunate, it is hard to change anything, once a font has been released and put into use for some time. It is usually necessary to create a new variant when you want to revise some design choices.

This is what happened to the STIX fonts, which were renamed to STIX Two after a major revision to the glyph shapes and some math alphabets.

Similarly, the New Computer Modern fonts can be considered a new variant of Latin Modern. While New Computer Modern can choose to disagree with Latin Modern and use different choices, any future revisions of Latin Modern will likely have to respect previous choices for compatibility.

4.1 Design choices of sans-serif

When choosing a sans-serif font for use in math, it is important to keep in mind that math alphabets are not meant for generic font switches, but for semantic markup of symbols in a formula. In physics, for example, bold sans-serif italic might be used for tensors, while bold italic might be used for vectors.

Besides providing a suitable range of Latin and Greek, the sans-serif glyphs also need to be clearly distinguishable from the corresponding serif glyphs based on their font properties, such as weight, width, contrast or stroke thickness.

While having some contrast between serif and sans-serif can be helpful, the sans-serif design should not be too incompatible with the main font, since the symbols from different alphabets should work together in a formula.

In general, it is better to combine serif and sans-serif fonts of similar weight and width, having just enough contrast in between to make them clearly distinguishable. It is also a good idea to use familiar shapes and to avoid any unusual shapes.

4.2 Design choices of Script/Calligraphic

When it comes to choices for Script or Calligraphic, there are two different styles how users expect a mathematical Script to look like.

The first group uses a restrained style of Script or Calligraphic. This includes the traditional styles used in Computer Modern, Euler Script, and Lucida Calligraphic:[3]

Neohellenic	ABCXYZ
Concrete	ABCXYZ
Garamond	ABCXYZ (StylisticSet=3)
KpFonts	ABCXYZ (StylisticSet=1)
XITS	ABCXYZ (StylisticSet=3)
Lucida	ABCXYZ (StylisticSet=4)
Euler	ABCXYZ
LM	ABCXYZ
New CM	ABCXYZabcxyz
STIX Two	ABCXYZabcxyz
Cambria	ABCXYZabcxyz
TG DejaVu	ABCXYZabcxyz

The second group uses a more fancy and elaborate style of formal Script. This includes the new design of Lucida Script:

Erewhon	ABCXYZ
XCharter	ABCXYZ
KpFonts	ABCXYZ
STIX Two	ABCXYZ (StylisticSet=1)
XITS	ABCXYZabcxyz
Libertinus	ABCXYZabcxyz
Garamond	ABCXYZabcxyz
Noto	ABCXYZabcxyz
TG Termes	ABCXYZabcxyz
TG Schola	ABCXYZabcxyz
Lucida	ABCXYZabcxyz

TEX Gyre Pagella uses a unique style, which could make this font's Script less usable:

TG Pagella	ABCXYZabcxyz

Several OpenType math fonts also provide an alternate style of Script or Calligraphic, which can be accessed using stylistic sets. These variants have also been included in the overview.

It is interesting to note that the STIX Two fonts have reversed a design decision of the XITS fonts regarding the choice of Script, and the designs have also been modified. New Computer Modern extends the Script from Latin Modern using the same style, while Concrete Math has adopted the original style of Calligraphic from Computer Modern.

[3] Some fonts have been scaled to match the size of other fonts: Lucida Calligraphic to 90% and Lucida Script to 85%, DejaVu to 90%, Termes, Pagella, and Schola to 95%.

4.3 Design choices of Fraktur/Blackletter

When it comes to choices for Fraktur or Blackletter, there is only one preferred style for how users expect a mathematical Fraktur to look.

The first group includes a majority of math font packages which use this typical style of Fraktur. Many fonts make use of Euler Fraktur, such as Latin Modern, New Computer Modern, and Pagella:[4]

LM	ABCXYZabcxyz
New CM	ABCXYZabcxyz
Concrete	ABCXYZabcxyz
Euler	ABCXYZabcxyz
Erewhon	ABCXYZabcxyz
XCharter	ABCXYZabcxyz
TG Pagella	ABCXYZabcxyz
TG Termes	ABCXYZabcxyz
Noto	ABCXYZabcxyz
Garamond	ABCXYZabcxyz
Cambria	ABCXYZabcxyz
Libertinus	ABCXYZabcxyz
STIX Two	ABCXYZabcxyz
XITS	ABCXYZabcxyz
TG Schola	ABCXYZabcxyz
TG DejaVu	ABCXYZabcxyz

The second group uses a Blackletter style instead of Fraktur, which is fairly unusual and could make these fonts less usable for this variant:

Neohellenic	ABCXYZ
Lucida	ABCXYZabcxyz
KpFonts	ABCXYZabcxyz

These designs could be just a fallback option when no suitable design of Fraktur was available.

4.4 Design choices of Blackboard Bold

When it comes to choices for Blackboard Bold, there are again two styles using a sans-serif or serif style of the BBold letters.

The first group uses a sans-serif style of BBold:

LM	ABCNOPQRXYZ abc 012
Euler	ABCNOPQRXYZ abc 012
Erewhon	ABCNOPQRXYZ abc 012
STIX Two	ABCNOPQRXYZ abc 012
XITS	ABCNOPQRXYZ abc 012
Lucida	ABCNOPQRXYZ
KpSans	ABCNOPQRXYZ
Neohellenic	ABCNOPQRXYZ
Fira	ABCNOPQRXYZ abc
Noto	ABCNOPQRXYZ abc 012
Lato	ABCNOPQRXYZ abc 012

[4] Some fonts have been scaled to match the size of other fonts: DejaVu to 90%, Schola to 95%. Termes, Pagella, and Lucida Blackletter are not scaled and shown at 100%.

The second group uses a serif style of BBold:[5]

New CM	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚
Concrete	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚
XCharter	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚
KpRoman	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ
Garamond	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚
Libertinus	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚
Cambria	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚
TG Schola	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚
TG Termes	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚
TG Pagella	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚
TG DejaVu	𝔸𝔹ℂℕ𝕆ℙℚℝ𝕏𝕐ℤ 𝕒𝕓𝕔 𝟘𝟙𝟚

While Latin Modern has adopted a sans-serif BBold, which also includes lowercase and numerals, New Computer Modern and Concrete Math have reverted to the traditional style of BBold from AMS fonts, at least for the uppercase. Many other fonts have chosen a scaled or adjusted variant of the sans-serif BBold from STIX/XITS.

5 Summary and conclusions

OpenType math fonts were introduced more than 15 years ago. Over the years, more math fonts have been developed and added to the font collection. As of this year, we have more than 20 choices of OpenType math fonts available (not counting variants) and more than 30 individual fonts (including variants and additional weights).

Nearly all OpenType math fonts discussed in this paper are free and readily available from CTAN or TeX Live, except for some non-free fonts and some unfinished projects from Github.

The available choices of OpenType math fonts cover most of what was previously available in other formats, including traditional TeX fonts (Computer Modern, Concrete, Euler), standard PostScript fonts (Times, Palatino), and other free PostScript fonts (Garamond, Utopia, Charter, DejaVu).

In our analysis, we have analyzed the coverage of math symbols and alphabets, as well as design choices and available font features.

While the range of symbols and alphabets may vary for each font, most available fonts will be good enough for general use, providing at least as much as traditional TeX fonts or even more.

Regarding design choices, most available font packages follow typical styles of how users expect mathematical Script, Fraktur, or Blackboard Bold to look. There are only few exceptions which use a unique or unusual style.

[5] Some fonts have been scaled to match the size of other fonts: DejaVu to 85%, Termes, Pagella, and Schola to 90%. Lucida is not scaled and shown at 100%.

In general, OpenType math fonts are not expected to provide the same level of stability and compatibility as traditional TeX fonts. While it should be possible to reprocess existing documents, you cannot expect the exact same line breaks, unless you archive and use specific versions of fonts.

In some cases, OpenType math fonts happen to be stable simply because they haven't been updated for years, but they may still exhibit the same bugs or limitations. Over time, it becomes more and more difficult to change anything. The longer a font has been left unchanged, the more likely that it will be necessary to introduce new variants for major revisions.

While font development is ongoing, OpenType math fonts are readily available for use today.

References

[1] Murray Sargent: High-quality editing and display of mathematical text in Office 2007. `https://learn.microsoft.com/en-us/archive/blogs/murrays`

[2] Microsoft Typography: OpenType specification, version 1.9, December 2021. `https://learn.microsoft.com/en-us/typography/opentype/spec`

[3] Ulrik Vieth: Do we need a Cork math font encoding? *TUGboat*, 29(3), 426–434, 2008. `https://tug.org/TUGboat/tb29-3/tb93vieth.pdf`
Reprinted in *MAPS*, 38, 3–11, 2009. `https://ntg.nl/maps/38/02.pdf`

[4] Ulrik Vieth: OpenType Math Illuminated. *TUGboat*, 30(1), 22–31, 2009. `https://tug.org/TUGboat/tb30-1/tb94vieth.pdf`
Reprinted in *MAPS*, 38, 12–21, 2009. `https://ntg.nl/maps/38/03.pdf`

[5] Jonathan Kew: XeTeX Live. *TUGboat*, 29(1), 151–156, 2008. `https://tug.org/TUGboat/tb29-1/tb91kew.pdf`

[6] Taco Hoekwater: Math in LuaTeX 0.40. *MAPS*, 38, 22–31, 2009. `https://ntg.nl/maps/38/04.pdf`

[7] Hans Hagen: LuaTeX math enhancements. *TUGboat*, 37(3), 269–274, 2016. `https://tug.org/TUGboat/tb37-3/tb117hagen-otmath.pdf`

[8] Tiro Typeworks: Cambria Math. `https://tiro.com/projects.html`

[9] Apostolos Syropoulos: Asana Math. `https://ctan.org/pkg/asana-math`

[10] Khaled Hosny: XITS font package.
https://ctan.org/pkg/xits
https://github.com/alif-type/xits

[11] STIX Consortium: STIX font package.
https://ctan.org/pkg/stix

[12] STIX Consortium: STIX2 font package.
https://ctan.org/pkg/stix2-otf
https://github.com/stipub/stixfonts

[13] GUST e-foundry: Latin Modern Math.
https://ctan.org/pkg/lm-math
https://gust.org.pl/projects/e-foundry

[14] GUST e-foundry: TEX Gyre Math.
https://ctan.org/pkg/tex-gyre-math
https://gust.org.pl/projects/e-foundry

[15] Bogusław Jackowski, Piotr Strzelczyk, Piotr Pianowski: GUST e-foundry font projects. *TUGboat*, 37(3), 269–274, 2016.
https://tug.org/TUGboat/tb37-3/tb117jackowski.pdf

[16] Bogusław Jackowski, Piotr Strzelczyk, Piotr Pianowski: Parametric math symbol fonts. *TUGboat*, 38(2), 208–211, 2017.
https://tug.org/TUGboat/tb38-2/tb119jackowski.pdf

[17] TEX Users Group: Lucida fonts from TUG.
https://tug.org/lucida/

[18] Khaled Hosny: Libertinus Fonts.
https://ctan.org/pkg/libertinus-fonts
https://github.com/alerque/libertinus

[19] Yuansheng Zhao, Xiangdong Zeng: Garamond Math.
https://ctan.org/pkg/garamond-math
https://github.com/YuanshengZhao/Garamond-Math

[20] Daniel Flipo: Erewhon Math.
https://ctan.org/pkg/erewhon-math

[21] Daniel Flipo: XCharter Math.
https://ctan.org/pkg/xcharter-math

[22] Daniel Flipo: KpFonts OTF package.
https://ctan.org/pkg/kpfonts-otf

[23] Antonis Tsolomitis: GFS Neohellenic Math.
https://ctan.org/pkg/gfsneohellenicmath

[24] Xiangdong Zeng: Fira Math.
https://ctan.org/pkg/firamath
https://github.com/firamath/firamath

[25] Chenjing Bu: Lato Math.
https://github.com/abccsss/LatoMath

[26] Łukasz Dziedzic: Lato Fonts.
https://github.com/latofonts/lato-source

[27] Noto Fonts Project: Noto Math.
https://github.com/notofonts/math

[28] Antonis Tsolomitis: New Computer Modern font family. *TUGboat*, 42(1), 52–55, 2021.
https://tug.org/TUGboat/tb42-1/tb130tsolomitis-newcm.pdf

[29] Antonis Tsolomitis: New Computer Modern.
https://ctan.org/pkg/newcomputermodern

[30] Daniel Flipo: Concrete Math.
https://ctan.org/pkg/concmath-otf

[31] Daniel Flipo: Euler Math.
https://ctan.org/pkg/euler-math

[32] Khaled Hosny: Neo Euler — An abandoned OpenType port of Euler math font.
https://github.com/aliftype/euler-otf

[33] Hans Hagen, Taco Hoekwater, Volker Schaa: Reshaping Euler: A collaboration with Hermann Zapf.
TUGboat, 29(3), 283–287, 2008.
https://tug.org/TUGboat/tb29-2/tb92hagen-euler.pdf

[34] Ulrik Vieth: OpenType math font development: Progress and challenges. *TUGboat*, 33(3), 302–308, 2012.
https://tug.org/TUGboat/tb33-3/tb105vieth.pdf

[35] Hans Hagen, Mikael P. Sundqvist: Pushing math forward with ConTEXt LMTX. *TUGboat*, 43(2), 202–206, 2022.
https://tug.org/TUGboat/tb43-2/tb134hagen-math.pdf

[36] Hans Hagen, Mikael P. Sundqvist: New directions in math fonts. *TUGboat*, 43(3), 300–310, 2022.
https://tug.org/TUGboat/tb43-3/tb135hagen-mathchange.pdf

[37] Hans Hagen, Mikael P. Sundqvist: Patching Lucida Bright Math. *TUGboat*, 43(3), 311–316, 2022.
https://tug.org/TUGboat/tb43-3/tb135hagen-lucida.pdf

[38] Frank Mittelbach: The `unicodefonttable` package. *TUGboat*, 42(3),287–304, 2021.
https://tug.org/TUGboat/tb42-3/tb132mitt-unicodefonttable.pdf

[39] Frank Mittelbach: `unicodefonttable`.
https://ctan.org/pkg/unicodefonttable

[40] Will Robertson: `unicode-math` package.
https://ctan.org/pkg/unicode-math

◇ Ulrik Vieth
Stuttgart, Germany
ulrik dot vieth (at) arcor dot de

Appendix: Font samples

As requested by conference participants, a selection of font samples has been included as an appendix.

The font samples show some typical equations from theoretical physics, using the proper notation conventions, such as bold italic for vectors.

These examples only use the main alphabets and don't use any Script, Fraktur or BBold. A much wider variety of examples would be needed to cover other notations. Other resources often use mathematical theorems in font samples, which may appear differently since they focus on other material.

Cambria and Lucida fonts

This section shows font samples for Cambria Math [8] and Lucida Bright Math [17], both of which are unique designs and also not free. Since Lucida tends to run larger and wider, the font samples had to be reduced to fit the column width.

Cambria Math:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Lucida Bright Math (95%):

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Variants of traditional TEX fonts

This section shows samples for variants of traditional TEX fonts, derived from Computer Modern, such as Latin Modern [13], New Computer Modern [29], and Concrete Math [30]. It also includes font samples for Euler Math [31] using an upright style.

Latin Modern (regular):

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

New Computer Modern (book):

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Concrete Math:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Euler Math:

$$\Delta\mathbf{E} - \frac{1}{c^2}\frac{\partial^2\mathbf{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\mathbf{j}}{\partial t}$$

$$\Delta\mathbf{B} - \frac{1}{c^2}\frac{\partial^2\mathbf{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\mathbf{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\mathbf{A}(\mathbf{r})\right)^2\psi + q\phi(\mathbf{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Variants of fonts derived from Times

This section shows samples of variants of traditional PostScript fonts derived from Times, such as TEX Gyre Termes [14], XITS (based on STIX 1.0) [10], and STIX Two [12].

While TEX Gyre and XITS follow the traditional look of Times, STIX Two includes a comprehensive redesign of the letter shapes for improved readability, which looks clearly different.

Further information about the redesign can be found in the package documentation, but it should be clear that STIX Two has been much improved, although it deviates from the traditional look.

TEX Gyre Termes:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

XITS (based on STIX 1.0):

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

STIX Two:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Variants of fonts derived from Palatino

This section shows samples of variants of traditional PostScript fonts derived from Palatino, such as TEX Gyre Pagella [14], KpRoman [22] derived from URW Palladio, and Asana Math [9].

While the letter shapes should be expected to be similar, there are significant differences in the design, sizing, and spacing of delimiters, and also in the placement of superscripts and subscripts.

Besides the differences in quality, KpFonts also provides a choice of additional weights, but only the regular version is shown here for comparison.

TEX Gyre Pagella:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

KpRoman Math (regular):

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Asana Math:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Other choices of serif fonts

This section shows font samples of other serif fonts, derived from traditional PostScript fonts or other freely available fonts. These include several of the TeX Gyre fonts [14], such as Schola, Bonum, and DejaVu, as well as other fonts such as Libertinus [18], Garamond Math [19], Erewhon, and XCharter Math [20, 21].

Since the Schola, Bonum, DejaVu and XCharter designs tend to run larger and wider, several font samples had to be reduced to fit the column width.

Along with the scaling also comes a reduction in height, which may not always be desirable.

TeX Gyre Schola (95%):

$$\Delta \boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial \boldsymbol{j}}{\partial t}$$

$$\Delta \boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{B}}{\partial t^2} = -\mu_0 \boldsymbol{\nabla} \times \boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial \psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2 \psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0 c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

TeX Gyre Bonum (95%):

$$\Delta \boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial \boldsymbol{j}}{\partial t}$$

$$\Delta \boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{B}}{\partial t^2} = -\mu_0 \boldsymbol{\nabla} \times \boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial \psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2 \psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0 c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

TeX Gyre DejaVu (95%):

$$\Delta \boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial \boldsymbol{j}}{\partial t}$$

$$\Delta \boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{B}}{\partial t^2} = -\mu_0 \boldsymbol{\nabla} \times \boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial \psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2 \psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0 c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Libertinus Math:

$$\Delta \boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial \boldsymbol{j}}{\partial t}$$

$$\Delta \boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{B}}{\partial t^2} = -\mu_0 \boldsymbol{\nabla} \times \boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial \psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2 \psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0 c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Garamond Math:

$$\Delta \boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial \boldsymbol{j}}{\partial t}$$

$$\Delta \boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{B}}{\partial t^2} = -\mu_0 \boldsymbol{\nabla} \times \boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial \psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2 \psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0 c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Erewhon Math (Utopia):

$$\Delta \boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial \boldsymbol{j}}{\partial t}$$

$$\Delta \boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{B}}{\partial t^2} = -\mu_0 \boldsymbol{\nabla} \times \boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial \psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2 \psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0 c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

XCharter Math (95%):

$$\Delta \boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial \boldsymbol{j}}{\partial t}$$

$$\Delta \boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2 \boldsymbol{B}}{\partial t^2} = -\mu_0 \boldsymbol{\nabla} \times \boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial \psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2 \psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0 c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Choices of sans-serif fonts

This section shows font samples of sans-serif fonts, derived from freely available fonts. These include KpSans [22], Neohellenic Math [23], Fira Math [24], Lato Math [25] (unsupported), and Noto Math [27]. Unfortunately, it isn't known which sans-serif design was chosen in the KpFonts distribution or from where it originates.

At the moment, Noto Math uses an inconsistent setup of sans-serif and serif alphabets, which has been partially corrected by substitutions in the example, but this isn't always possible.

KpSans Math:

$$\Delta E - \frac{1}{c^2}\frac{\partial^2 E}{\partial t^2} = \frac{1}{\varepsilon_0}\nabla\lambda + \mu_0\frac{\partial j}{\partial t}$$

$$\Delta B - \frac{1}{c^2}\frac{\partial^2 B}{\partial t^2} = -\mu_0\nabla\times j$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\nabla - qA(r)\right)^2\psi + q\phi(r)\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Neohellenic Math:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Lato Math:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Fira Math:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Noto Math:

$$\Delta\boldsymbol{E} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{E}}{\partial t^2} = \frac{1}{\varepsilon_0}\boldsymbol{\nabla}\lambda + \mu_0\frac{\partial\boldsymbol{j}}{\partial t}$$

$$\Delta\boldsymbol{B} - \frac{1}{c^2}\frac{\partial^2\boldsymbol{B}}{\partial t^2} = -\mu_0\boldsymbol{\nabla}\times\boldsymbol{j}$$

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \frac{1}{2m}\left(\frac{\hbar}{\mathrm{i}}\boldsymbol{\nabla} - q\boldsymbol{A}(\boldsymbol{r})\right)^2\psi + q\phi(\boldsymbol{r})\,\psi$$

$$\gamma^\alpha\left(\frac{\hbar}{\mathrm{i}}\partial_\alpha - qA_\alpha\right)\psi + m_0c\,\psi = 0$$

$$R^{\mu\nu} - \frac{1}{2}Rg^{\mu\nu} + \Lambda g^{\mu\nu} = -\frac{8\pi G}{c^2}M^{\mu\nu}$$

Standardizing OpenType math fonts

Hans Hagen, Mikael P. Sundqvist

1 Introduction

ConTEXt has always had a good support for the typesetting of mathematics. ConTEXt MkII uses the pdfTEX engine and hence traditional (Type 1) fonts. Several math fonts are available, specifically designed to work seamlessly with TEX. ConTEXt MkIV, the successor version, utilizes the LuaTEX engine, providing support not only for traditional fonts but also for OpenType Unicode math fonts. Unlike the XƎTEX engine, which interpreted these new fonts in a manner similar to traditional TEX fonts, LuaTEX adheres more closely to the (unfortunately somewhat vague) OpenType specification.[1] When new fonts appeared, some were more like the traditional fonts, others more like OpenType Unicode math fonts. This leads to difficulties in achieving consistent results across different fonts and might be one reason that the Unicode engines are not yet used as much as they probably should.

In autumn 2021 we started to discuss how to improve the typesetting of OpenType Unicode mathematics, and it was natural to go on and do this for the LuaMetaTEX engine, and hence for ConTEXt LMTX. Since then, we have been engaging in daily discussions covering finer details such as glyphs, kerning, accent placement, inter-atom spacing (what we refer to as math microtypography), as well as broader aspects like formula alignment and formula line breaking (math macrotypography). This article will primarily focus on the finer details. Specifically, we will explore the various choices we have made throughout the process. The OpenType Unicode math specification is incomplete; some aspects are missing, while others remain ambiguous. This issue is exacerbated by the varying behaviors of fonts.

We make runtime changes to fonts, and add a few additional font parameters that we missed. As a result, we deviate from the standard set by Microsoft (or rather, we choose to interpret it in our own way) and exercise the freedom to make runtime changes to font parameters. Regarding this aspect, we firmly believe that our results often align more closely with the original intentions of the font designers. Indeed, the existence of “oddities” in these fonts may be attributed to the lack of an engine, during their creation, that supported all the various features, making testing difficult, if not essentially impossible. Within ConTEXt LMTX, we have the necessary support, and we can activate various helpers that allow us to closely examine formulas. Without them our work would not have been possible.

Ultimately, we hope and believe that we have made straightforward yet effective choices, rendering the existing OpenType Unicode math fonts usable. We hope that this article might be inspiring and useful for others who aim to achieve well-designed, modern math typesetting.

2 Traditional vs. OpenType math fonts

There is a fundamental difference between traditional TEX math fonts and OpenType Unicode fonts. In the traditional approach, a math setup consists of multiple independent fonts. There is no direct relationship between a math italic x and an $\hat{}$ on top of it. The engine handles the positioning almost independently of the shapes involved. There can be a shift to the right of $\hat{x}$ triggered by kerning with a so-called skew character but that is it.

A somewhat loose coupling between fonts is present when we go from a base character to a larger variant that itself can point to a larger one and eventually end up at an extensible recipe. But the base character and that sequence are normally from different fonts. The assumption is that they are designed as a combination. In an OpenType font, variants and extensibles more directly relate to a base character.

Then there is the italic correction which adds kerns between a character and what follows depending on the situation. It is not, in fact, a true italic correction, but more a hack where an untrue width is compensated for. A traditional TEX engine defaults to adding these corrections and selectively removes or compensates for them. In traditional TEX this fake width helps placing the subscript properly while the italic correction is added to the advance width when attaching subscripts and/or moving to the next atom.

In an OpenType font we see these phenomena translated into features. Instead of many math fonts we have one font. This means that one can have relations between glyphs, although in practice little of that happens. One example is that a specific character can have script and scriptscript sizes with a somewhat different design. Another is that there can be alternate shapes for the same character, and yet another is substitution of (for instance) dotted characters by dotless ones. However, from the perspective of features a math font is rather simple and undemanding.

Another property is that in an OpenType math font the real widths are used in combination with

[1] learn.microsoft.com/en-us/typography/opentype/spec/math

doi.org/10.47397/tb/44-2/tb137sundqvist-otmath

optional italic correction when a sequence of characters is considered text, with the exception of large operators where italic correction is used for positioning limits on top and below. Instead of abusing italic corrections this way, a system of staircase kerns in each corner of a shape is possible.

Then there are top (but not bottom) anchor positions that, like marks in text fonts, can be used to position accents on top of base characters or boxes. And while we talk of accents: they can come with so-called flat substitutions for situations where we want less height.

All this is driven by a bunch of font parameters that (supposedly) relate to the design of the font. Some of them concern rules that are being used in constructing, for instance, fractions and radicals but maybe also for making new glyphs like extensibles, which is essentially a traditional TEX thing.

So, when we now look back at the traditional approach we can say that there are differences in the way a font is set up: widths and italic corrections, staircase kerns, rules as elements for constructing glyphs, anchoring of accents, flattening of accents, replacement of dotted characters, selection of smaller sizes, and font parameters. These differences have been reflected in the way engines (seem to) deal with OpenType math: one can start with a traditional engine and map OpenType onto that; one can implement an OpenType engine and, if needed, map traditional fonts onto the way that works; and of course there can be some mix of these.

In practice, when we look at existing fonts, there is only one reference and that is Cambria. When mapped onto a traditional engine, much can be made to work, but not all. Then there are fonts that originate in the TEX community and these do not always work well with an OpenType engine. Other fonts are a mix and work more or less. The more one looks into details, the clearer it becomes that no font is perfect and that it is hard to make an engine work well with them. In LuaMetaTEX we can explicitly control many of the choices the math engine makes, and there are more such choices than with traditional TEX machinery. And although we can adapt fonts at runtime to suit the possibilities, it is not pretty.

This is why we gradually decided on a somewhat different approach: we use the advantage of having a single font, normalize fonts to what we can reliably support, and if needed, add to fonts and control the math engine, especially the various subsystems, with directives that tell it what we want to be done. Let us discuss a few things that we do when we load a math font.

3 Getting rid of italic corrections

OpenType math has italic corrections for using characters in text and large operators (for limits), staircase kerns for combining scripts, and top anchor for placement of accents. In LuaMetaTEX we have access to more features.

Let's remind ourselves. In a bit more detail, OpenType has:

- `An italic correction` is injected between characters in running text, but: a sequence of atoms is *not* text, they are individually spaced.
- An `italic correction` value in large operators that reflects where limits are attached in display mode; in effect, using the italic correction as an anchor.
- `Top anchors` are used to position accents over characters, but not so much over atoms that are composed from more than characters, e.g., including fractions, fences, radicals, and so on.
- `Flat accents` as substitution feature for situations where the height would become excessive.
- `Script and scriptscript` as substitution feature for a selection of characters that are sensitive for scaling down.

This somewhat limited view on math character positioning has been extended in LuaMetaTEX, and we remap the above onto what we consider a bit more reliable, especially because we can tweak these better. We have:

- `Corner kerns` that make it possible to adjust the horizontal location of sub- and superscripts and prescripts.
- Although `flat accents` are an existing feature, we extended them by providing additional scaling when they are not specified.
- In addition to script sizes we also have `mirror` as a feature so that we can provide right to left math typesetting. (This also relates to dropping in characters from other fonts, like Arabic.)
- In addition to the `top anchors` we also have `bottom anchors` in order to properly place bottom accents. These are often missing, so we need to construct them from available snippets.
- An additional `extensible italic correction` makes it possible to better anchor scripts to sloped large operators. This is combined with keeping track of `corner kerns` that can be specified per character.
- Characters can have `margins` which makes it possible to more precisely position accents that would normally overflow the base character and clash with scripts. These go in all four directions.

- In order to be able to place the degree in a radical more precisely (read: not run into the shape when there is more than just a single degree atom) we have `radical offsets`.

There are plenty more tuning options but some are too obscure to mention here. All high level constructors, like fences, radicals, accents, operators, fractions, etc. can be tuned via optional keyword and key/values at the macro end.

We eliminate the italic correction in math fonts, instead adding it to the width, and using a negative bottom right kern. If possible we also set a top and bottom accent anchor. This happens when we load the font. We also translate the italic correction on large operators into anchors. As a result, the engine can now completely ignore italic corrections in favor of proper widths, kerns and anchors. Let us look at a few examples.

The italic f is used a lot in mathematics and it is also one of the most problematic characters. In TeX Gyre Bonum Math the italic f has a narrow bounding box; the character sticks out on both the left and right. To the right, this is compensated by a large amount of italic correction. This means that when one adds sub- and superscripts, it works well. We add italic correction to the width, and introducing a negative corner kern at the bottom right corner, and thus the placement of sub- and superscripts is not altered. Look carefully at the bounding boxes below.

$$f_0^1 \qquad f_0^1$$

original tweaked

Compare with Lucida Bright Math, which comes with staircase kerns instead of italic correction. We convert these kerns into corner kerns.

$$f_0^1 \qquad f_0^1$$

original tweaked

For characters that stick out to the left, we also increase the width and shift the glyph to ensure that it does not stick out on the left side. This prevents glyphs from clashing into each other.

$$(f) \qquad (f)$$

original tweaked

As mentioned, for the integral, one of the most common big operators, the limits are also placed with help of the italic correction. When the limits go below and on top, proper bottom and top anchor points are introduced, calculated from the italic correction. (The difference in size of the integral signs is a side effect of the font parameter `DisplayOperatorMinHeight` being tweaked, as we'll discuss more later. OpenType fonts can come with more than two sizes.)

$$\int_a^b f(x)\,dx \qquad \int_a^b f(x)\,dx \qquad \int\limits_a^b f(x)\,dx$$

original tweaked, nolimits tweaked, limits

Compare these integrals with the summation, that usually does not have any italic correction bound to it. This means that the new anchor points end up in the middle of the summation symbol.

$$\sum_{k=1}^{n} a_k \qquad \sum_{k=1}^{n} a_k$$

original tweaked

We also introduce some corner kerns in cases where there were neither italic corrections nor staircase kerns. This is mainly done for delimiters, like parentheses. We can have a different amount of kerning for the various sizes. Often the original glyph does not benefit from any kerning, while the variants and extensibles do.

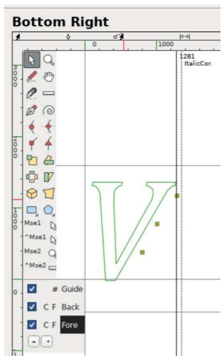
$$\left(\frac{1}{1+x^2}\right)^2 \qquad \left(\frac{1}{1+x^2}\right)^2$$

original tweaked

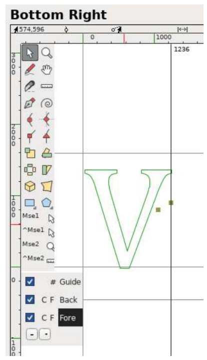
Note also the different sizes of the parentheses in the example above. Both examples are set with `\left(` and `\right)`, but the font parameters are chosen differently in the tweaked version. Font designers should have used the opportunity to have more granularity in sizes. Latin Modern Math has four, many others have steps in between, but there is a lack of consistency.

4 Converting staircase kerns

We simplify the staircase kerns, which are often somewhat sloppy and seldom complete (see figure below), into more reliable corner kerns. It's good enough and looks better on the whole. We also avoid bugs that way.



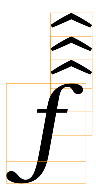
italic V



upright V

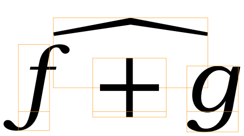
5 Tweaking accents

We ignore the zero dimensions of accents, simply assuming that one cannot know if the shape is centered or sticks out in a curious way, and therefore use proper widths with top and bottom anchors derived from the bounding box. We compensate for negative llx values being abused for positioning. We check for overflows in the engine. In case of multiple accents, we place the first one anchored over the character, and center the others on top of it.

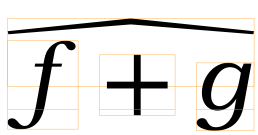


We mentioned in an earlier *TUGboat* article[2] that sometimes anchor points are just wrong. We have a tweak that resets them (to the middle) that we use for several fonts and alphabets.

Some accents, like the hat, can benefit from being scaled. The fonts typically provide the base size and a few variants.

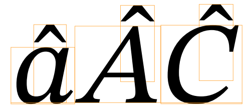


original

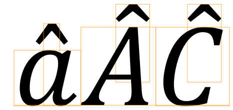


tweaked

The only fonts we have seen that support flattened accents are Stix Two Math and Cambria Math.



Stix Two



Cambria

If you look carefully, you notice that the hats over the capital letters are not as tall as the one over the lowercase letter. There is a font parameter `FlattenedAccentBaseHeight` that is supposed to specify when this effect is supposed to kick in. Even though other fonts do not use this feature, the parameter is set, sometimes to strange values (if they were to have the property). For example, n Garamond Math, the value is 420.

We introduced a tweak that can fake the flattened accents, and therefore we need to alter the value of the font parameter to more reasonable values. We communicated to Daniel Flipo, who maintains several math fonts, that the parameter was not correctly set in Erewhon math. In fact, it was set such that the flattened accents were used for some capital letters (C in the example below) but not for others (A below). He quickly fixed that. The green (gray in print) rules in the picture have the height of `FlattenedAccentBaseHeight`; it did not need to be decreased by much.

Erewhon, not fixed

Erewhon, fixed

6 Getting rid of rules

We get rid of rules as pseudo-glyphs in extensibles and bars. This also gives nicer visual integration because flat rules do not always fit in with the rest of the font. We also added support for this in the few (Polish) Type 1 math fonts that we still want to support, like Antykwa Toruńska.

$$\sqrt{\frac{1+x}{1-x}}$$

with rule

$$\sqrt{\frac{1+x}{1-x}}$$

with glyph

$$\sqrt{\frac{1+x}{1-x}}$$

Antykwa

Here is an enlarged example of an Antykwa rule. Latin Modern has rounded corners, here we see a rather distinctive ending.

$$\overline{x^2+2x+2}$$

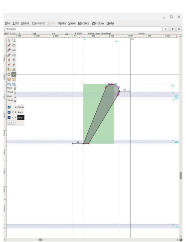
[2] "New directions in math fonts", **43**:3, `tug.org/TUGboat/tb43-3/tb135hagen-mathchange.pdf`

7 Tweaking primes

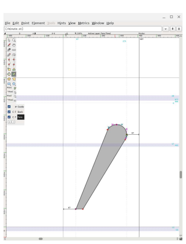
We make it no secret that we consider primes in math fonts a mess. For some reason no one could convince the Unicode people that a 'prime' is not a 'minute' (that is, U+2032 PRIME is also supposed to be used as the symbol for minutes); in case you'd like to argue that "they often look the same", that is also true for the Latin and Greek capital 'A'. This lost opportunity means that, as with traditional TEX fonts, we need to fight a bit with placement. The base character can or cannot be already anchored at some superscript-like position, so that makes it basically unusable. An alternative assumption might be that one should just use the script size variant as a superscript, but as we will see below, that assumes that they sit on the baseline so that we can move it up to the right spot. Add to that the fact that traditional TEX has no concept of a prime, and we need some kind of juggling with successive scripts. This is what macro packages end up doing.

But this is not what we want. In ConTEXt MkIV we already have special mechanisms for dealing with primes, which include mapping successive primes onto the multiple characters in Unicode, where we actually have individual triple and quadruple primes and three reverse (real) primes as well. However, primes are now a native feature, like super- and subscripts, as well as prescripts and indices. (All examples here are uniformly scaled.)

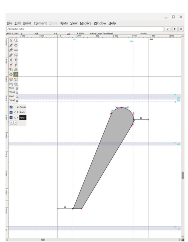
Because primes are now a native feature, we also have new font parameters `PrimeShiftUp` and `PrimeShiftUpCramped`, similar to `SuperscriptShiftUp` and `SuperscriptShiftUpCramped`, which add a horizontal axis where the primes are placed. There is also a `fixprimes` tweak that we can use to scale and fix the glyph itself. Below, we see how very different the primes from different fonts look (all examples are uniformly scaled), and then examples comparing the original and tweaked primes.



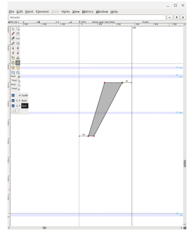
Latin Modern



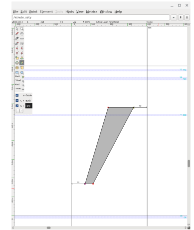
st



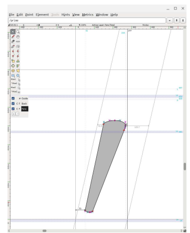
sts



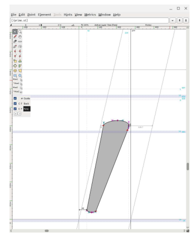
Lucida



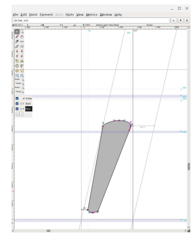
ssty



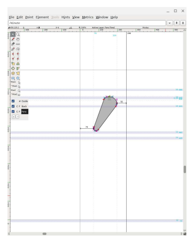
Erewhon



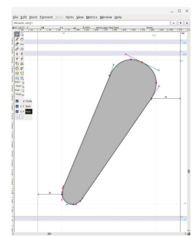
st



sst



Libertinus



ssty1

$f'(x) + e^{f'(x)}$

Latin Modern original

$f'(x) + e^{f'(x)}$

tweaked

$f'(x) + e^{f'(x)}$

Lucida original

$f'(x) + e^{f'(x)}$

tweaked

$f'(x) + e^{f'(x)}$

Erewhon original

$f'(x) + e^{f'(x)}$

tweaked

$f'(x) + e^{f'(x)}$

Libertinus original

$f'(x) + e^{f'(x)}$

tweaked

8 Font parameters

We add some font parameters, ignore some existing ones, and fix at runtime those that look to be sub-optimal. We have no better method than looking at examples, so parameters might be fine-tuned further in the future. Following are examples of pdfLATEX math, LuaLATEX math, and (as of this writing) ConTEXt LMTX:

$$h^3 + h_2 + h_2^3 + h'$$

$$h^3 + h_2 + h_2^3 + h'$$

$$h^3 + h_2 + h_2^3 + h'$$

We have already mentioned that we have a few new parameters, `PrimeShiftUp` and `PrimeShiftUpCramped`, to position primes on their own axis, independent of the superscripts. They are also chosen to always be placed outside superscripts, so the inputs `$f'^2$` and `$f^2'$` both result in $f^{2\prime}$. Authors should use parentheses in order to avoid confusion.

Let us briefly mention the other parameters. These are the adapted parameters for TeX Gyre Bonum:

```
AccentTopShiftUp                  =  -15
FlattenedAccentTopShiftUp         =  -15
AccentBaseDepth                   =   50
DelimiterPercent                  =   90
DelimiterShortfall                =  400
DisplayOperatorMinHeight          = 1900
SubscriptShiftDown                =  201
SuperscriptShiftUp                =  364
SubscriptShiftDownWithSuperscript
  = "1.4*SubscriptShiftDown"
PrimeShiftUp
  = "1.25*SuperscriptShiftUp"
PrimeShiftUpCramped
  = "1.25*SuperscriptShiftUp"
```

Some of these are not in OpenType. We can set up much more, but it depends on the font what is needed, and also on user demands.

We have noticed that many font designers seem to have had problems setting some of the values; for example, `DisplayOperatorMinHeight` seems to be off in many fonts.

9 Profiling

Let us end with profiling, which is only indirectly related to the tweaking of the fonts. Indeed, font parameters control the vertical positioning of sub- and superscripts. If not carefully set, they might force a non-negative `\lineskip` where not necessary. In the previous section we showed how these parameters were tweaked for Bonum.

Sometimes formulas are too high (or have a too large depth) for the line, and so a `\lineskip` is added so that the lines do not clash. If the lowest part of the top line (typically caused by the depth) and the tallest part of the bottom line (caused by the height) are not close to each other on the line, one might argue that this `\lineskip` does not have to be added, or at least with reduced amount. This is possible to achieve by adding `\setupalign[profile]`. An example is in figure 1.

In the figure, we enabled a helper that shows us where the profiling feature kicks in. We also show the lines (`\showmakeup[line]`). Below we show the example without those helpers. You can judge for yourself which one you prefer.

It is worth emphasizing that, contrary to what one might believe at first, the profiling does not substantially affect the compilation time. On a 300-page math book we tried, which usually compiles in about 10 seconds, profiling did not add more than 0.5 seconds. The same observation holds for the other math tweaks we have mentioned: the overhead is negligible.

10 Conclusions

All these tweaks can be overloaded per glyph if needed; for some fonts, we indeed do this, in so-called goodie files. The good news is that by doing all this we present the engine with a font that is consistent, which also means that we can more easily control the typeset result in specific circumstances.

The reader may wonder how we ended up with this somewhat confusing state of affairs in the font world. Here are some possible reasons. There is only one reference font, Cambria, and that uses its reference word processor renderer, Word. Then came XeTeX that as far as we know maps OpenType math onto a traditional TeX engine, so when fonts started coming from the TeX crowd, traditional dimensions and parameters sort of fit in. When LuaTeX showed up, it started from the other end: OpenType. That works well with the reference font but less so with that ones coming from TeX. Eventually more fonts showed up, and it's not clear how these got tested because some lean towards the traditional and others towards the reference fonts. And, all in all, these fonts mostly seem to be rather untested in real (more complex) math.

The more we looked into the specific properties of OpenType math fonts and rendering, the more we got the feeling that it was some hybrid of what TeX does (with fonts) and ultimately desired behavior. That works well with Cambria and a more or less frozen approach in a word processor, but doesn't suit well with TeX. Bits and pieces are missing, which could have been added from the perspective of generalization and imperfections in TeX as well. Lessons learned from decades of dealing with math in macros

So the question is: how good an approximation to σ is $\sigma * W\phi$? But the attentive reader will realize that we have already answered this question in the course of proving the sharp Gårding inequality. Indeed, suppose $\phi \in \mathcal{S}$ is even and $\|\phi\|_2 = 1$, and set $\phi^a(x) = a^{n/4}\phi(a^{1/2}x)$. Then we have shown (cf. Remark (2.89)) that $\sigma - \sigma * W\phi^a \in S^{m-(\rho-\delta)}_{\rho,\delta}$ whenever $\sigma \in S^m_{\rho,\delta}$ is supported in a set where $\langle\xi\rangle^{\rho+\delta} \approx a$.

No profiling

So the question is: how good an approximation to σ is $\sigma * W\phi$? But the attentive reader will realize that we have already answered this question in the course of proving the sharp Gårding inequality. Indeed, suppose $\phi \in \mathcal{S}$ is even and $\|\phi\|_2 = 1$, and set $\phi^a(x) = a^{n/4}\phi(a^{1/2}x)$. Then we have shown (cf. Remark (2.89)) that $\sigma - \sigma * W\phi^a \in S^{m-(\rho-\delta)}_{\rho,\delta}$ whenever $\sigma \in S^m_{\rho,\delta}$ is supported in a set where $\langle\xi\rangle^{\rho+\delta} \approx a$.

Profiling

So the question is: how good an approximation to σ is $\sigma * W\phi$? But the attentive reader will realize that we have already answered this question in the course of proving the sharp Gårding inequality. Indeed, suppose $\phi \in \mathcal{S}$ is even and $\|\phi\|_2 = 1$, and set $\phi^a(x) = a^{n/4}\phi(a^{1/2}x)$. Then we have shown (cf. Remark (2.89)) that $\sigma - \sigma * W\phi^a \in S^{m-(\rho-\delta)}_{\rho,\delta}$ whenever $\sigma \in S^m_{\rho,\delta}$ is supported in a set where $\langle\xi\rangle^{\rho+\delta} \approx a$.

No profiling

So the question is: how good an approximation to σ is $\sigma * W\phi$? But the attentive reader will realize that we have already answered this question in the course of proving the sharp Gårding inequality. Indeed, suppose $\phi \in \mathcal{S}$ is even and $\|\phi\|_2 = 1$, and set $\phi^a(x) = a^{n/4}\phi(a^{1/2}x)$. Then we have shown (cf. Remark (2.89)) that $\sigma - \sigma * W\phi^a \in S^{m-(\rho-\delta)}_{\rho,\delta}$ whenever $\sigma \in S^m_{\rho,\delta}$ is supported in a set where $\langle\xi\rangle^{\rho+\delta} \approx a$.

Profiling

Figure 1: Above: comparison of standard (no profiling) and math profiling typesetting, with guides for where profiling occurred; namely, the fourth and fifth baselines are altered. Below: the same, without the guides.

and math fonts were not reflected in the OpenType fonts and approach, which is of course understandable as OpenType math never especially aimed at TeX. But that also means that at some point one has to draw conclusions and make decisions — which is what we do in ConTeXt, LuaMetaTeX and the runtime-adapted fonts. And it gives pretty good and reliable results.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University

Behind the scenes of the Great Ti*k*Zlings Christmas Extravaganza

samcarter, Gert Fischer

"O, wonder! How many goodly creatures are there here."
(William Shakespeare, The Tempest, V, 1)

Abstract

The Great Ti*k*Zlings Christmas Extravaganza is an annual video series that utilises LaTeX to produce animated films. This proceeding will offer a look behind the scenes of the Extravaganza and explain how we use LaTeX to create the videos.

1 The Rise of the Ti*k*Zling

In 2005 Till Tantau was about to publish his new tool to create graphic elements in LaTeX and was looking for a name. He turned to a recursive acronym and chose "Ti*k*Z ist kein Zeichenprogramm", in English: "Ti*k*Z is not a drawing program". This might be understood as a cautioning of users not to expect too much. The instant success and the usefulness of the package and its updates till today show that this probably was too modest a name. Also, as in all great creations, there were hidden dimensions the creator had not intentionally included and could not and did not foresee.

So Ti*k*Z too had its Goedelian moments. One came when samcarter was probably the first human to discover that there were animals in Ti*k*Z. In 2017 she found a duck. And in quick succession other creatures were discovered. With the help of Paulo Cereda, Ulrike Fischer, Carla Maggi and many others, samcarter listed and categorised them and in a second step supplied them with clothes and other items helping them to feel comfortable at the fringes of Ti*k*Z. The question of how to name the newly discovered was put to the TikZ community and resolved democratically. In an open ballot the members voted almost unanimously for Stefan Kottwitz's proposal. From now on there would be "Ti*k*Zlings". By 2018 the Ti*k*Zlings ecosystem had been thoroughly documented on GitHub [3] in a flexible grid allowing the addition of hitherto unknown species.

While this important work was going on, some of samcarter's collaborators decided in 2017 to honour her efforts with a special Christmas present. They took it upon them to teach some of the Ti*k*Zducks to dance and sing. As the following quotes from their e-mail exchange show, Paulo Cereda, Gert Fischer, Ulrike Fischer and Carla Maggi were in deep water at once:

- "never tried it before"
- "I tried something but it doesn't work"
- "I have almost no experience with videos"
- "when everything fails, I'll record my own screen"
- "the arara rule doesn't work"
- "it's the first time I use github"

But somehow it worked. At the end the resulting memes were merged into a video film under the title "The Great Ti*k*Zducks Christmas Extravaganza". Samcarter loved it!

From then on there was no looking back. Samcarter joined the team bringing in the expertise so sorely lacking in the first Extravaganza. Two other additions were Professore Paulinho van Duck of Sempione Park University, Milano, Italy, and Bär who came in as assistant to the producers and fashion consultant to the Ti*k*Zlings cast [1]. And the Extravaganza was indeed there to stay. The Ti*k*Zducks Extravaganza became the Ti*k*Zlings Extravaganza which since 2018 has appeared every year in December, sometimes with the additional help of non-permanent team members such as Marmot and Plergux. All editions can be viewed on the Internet [2]. Over the years the Extravaganza has gained a group of fervent supporters and it has also made its impact on TUG, where the outgoing president Boris Veytsman has been recommending it to the members from the start.

Having said as much, a view behind the scenes proves that Christmas preparations require some programming skills — or as The Bard has it: *"Though this be madness, yet there is method in't"* (Hamlet, Prince of Denmark, II, 2).

2 Creating multi-page PDFs

The basic idea of creating animated films with LaTeX is to first create a multi-page PDF with incremental motion between each page, which then gets converted into a movie.

There are many possible ways of using LaTeX to create a multi-page PDF. For many of our scenes, we use a combination of beamer and Ti*k*Z.

The overlay mechanism of beamer is used to repeat the animation on all pages, and Ti*k*Z makes it easy to position individual elements on a page.

The following code block shows a shortened example of how this setup can be used to let a penguin move across a page:

```
\documentclass{beamer}
\usepackage{tikz}
\usetikzlibrary{tikzlings}
\begin{document}

\begin{frame}
```


doi.org/10.47397/tb/44-2/tb137samcarter-tikzlings

```
  \begin{tikzpicture}[remember picture,overlay]
    \foreach \macro in {1,...,5}{
        \path<+>[fill=white]
           (2*\macro,-0.7*\macro) pic{penguin};
    }
  \end{tikzpicture}
\end{frame}
\end{document}
```

This code produces a PDF with five pages. In Figure 1 the five pages are stacked on top of each other with different levels of transparency to visualise the movement of the penguin.

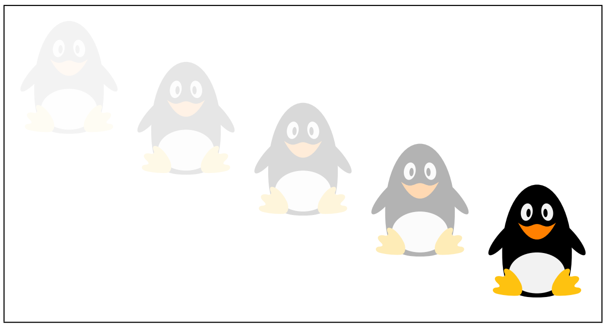


Figure 1: Visualisation of a penguin moving across a page. The different opacity levels denote the different pages stacked on top of each other. (Grayscaled for print.)

3 Converting PDFs into videos

Converting a PDF into a video consists of multiple steps. The first step is converting each page of the PDF into a raster image. We are using PNG as image format as many of the protagonists in our videos consist of simple geometric shapes with clear edges and PNG is a format which will preserve these edges and won't introduce image artefacts.

There are many tools available to convert PDFs into PNGs. Currently, our tool of choice is `pdftoppm`, which is part of the Poppler[1] package:

```
pdftoppm -png -r 240 Example.pdf Example
```

Compared to the widely used `convert` command from ImageMagick, we found that `pdftoppm` is faster in converting the PDF and lighter on CPU usage.

The next step is to use FFmpeg[2] to assemble the individual images into a video and combine it with suitable music:

```
ffmpeg \
  -ss 00:00:00 -i Example-%03d.png \
  -ss 00:00:10 -i Music.m4a \
  -shortest \
  Example_raw.mp4
```

We sometimes encountered problems with videos not working in some multimedia players, so we now preventively run the resulting video through HandBrake:[3]

```
HandBrakeCLI --crop 0:0:0:0 \
  -i Example_raw.mp4 -o Example.mp4
```

This leaves us with one video clip for each scene and each intermission, which need to be combined into a single video. To make the resulting video more pleasant to watch, transitions between the individual videos have to be inserted and the volume level of some of the clips needs to be adjusted to fit in with the others.

In the past, we used video editing applications like iMovie (macOS) to combine the videos. They usually required some level of manual interaction, e.g. dragging videos into the correct order, clipping them if necessary, etc. This turned out to be error-prone and could also be frustrating if there were last-minute changes to one of the clips which required re-rendering the whole video.

Our current solution is to script the whole process using `moviepy`,[4] a python library for video editing. An abbreviated example of our merge script is shown in the following:

```
###############################################
# stitching together the videos
# and adding transitions
###############################################
from moviepy.editor import *
import os

# duration of transitions between the videos
padding = 1.5

video_clips = [
VideoFileClip("../intermissions/title.mp4"),
VideoFileClip("../intermissions/example.mp4"),
VideoFileClip("../example-scene/Example.mp4")
↪ .volumex(1.1),
VideoFileClip("../intermissions/credits.mp4"),
]

###############################################
# merge title and first intermission to get
# continious audio
###############################################
video_fx_list = []
idx = 0
for video in video_clips[0:2]:
    video_fx_list.append(video.set_start(idx)
```

1 poppler.freedesktop.org/
2 ffmpeg.org/
3 handbrake.fr/
4 zulko.github.io/moviepy/

```
    ↪ .crossfadein(padding)
    ↪ .crossfadeout(padding)
    ↪ .audio_fadein(padding)
    ↪ .audio_fadeout(padding))
    idx += video.duration - padding

merged_video =
↪ CompositeVideoClip(video_fx_list)
duration = merged_video.end

audioclip = AudioFileClip("../intermissions/
↪ JingleBells.m4a").subclip(2,duration+2)
merged_video =
↪ merged_video.set_audio(audioclip)

###############################################
# adding rest of the scenes
###############################################
video_fx_list = []
idx = 0

# adding the merged video first
video_fx_list.append(merged_video
↪ .set_start(idx).crossfadein(padding)
↪ .crossfadeout(padding).audio_fadein(padding)
↪ .audio_fadeout(padding))
idx += merged_video.duration - padding

# rest of the videos
for video in video_clips[2:]:
    video_fx_list.append(video.set_start(idx)
    ↪ .crossfadein(padding)
    ↪ .crossfadeout(padding)
    ↪ .audio_fadein(padding)
    ↪ .audio_fadeout(padding))
    idx += video.duration - padding

final_video = CompositeVideoClip(video_fx_list)
final_video.write_videofile
↪ ("Extravaganza_raw.mp4")
```

The full version of this `moviepy` script is available from `github.com/TikZlings/Extravaganza2022/blob/main/videos/merge_videos.py`.

The final step is another pass through HandBrake. This will ensure that the video is properly encoded and all potential issues are fixed. It also reduces the file size quite a bit.

4 Watch the videos

The finished video gets published on Vimeo. If you would like to watch some of the previous videos, we collected the links to all our previous videos at `github.com/TikZlings`. This site also contains links to the source code of all the previous extravaganzas in case you are curious about how a particular clip was made.

References

[1] U. Fischer. *The Bearwear package: Shirts to dress TikZbears.* `ctan.org/pkg/bearwear`

[2] Ti*k*Zlings organisation, 2023. `github.com/TikZlings`

[3] samcarter. Ti*k*Zlings, 2023. `github.com/samcarter/tikzlings`

◇ samcarter

◇ Gert Fischer
Bonn, Germany

Curvature combs and harmonized paths in METAPOST

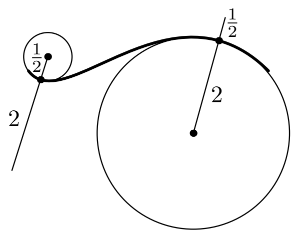
Linus Romer

Abstract

Most font editors offer curvature-related tools. One of these tools is the visualization of curvature via *curvature combs*. Another tool is the so-called *harmonization*, which makes the curvature continuous along paths. An implementation of both tools in METAPOST will be presented. Curvature-optimized paths already play a significant role in METAFONT and METAPOST and therefore example METAPOST paths will be examined for their curvature behavior.

1 Curvature

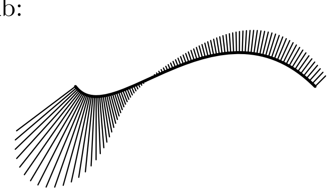
The curvature in a point of a curve is the inverse of the radius of the osculating circle at this point (depicted here as a "curvature vector" on the opposite side of the radius):



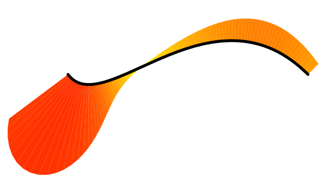
For straight segments, the curvature is constantly zero, since the radius of the osculating circle is infinitely large. Vice versa, the curvature becomes infinitely large when the radius of the osculating circle tends to zero.

2 Curvature combs in METAPOST

We can assemble these curvature vectors into a curvature comb:



The curvature may be additionally mapped to a color and the gaps may be filled (all images are grayscaled for print):



A figure like the above can be achieved with

`path p; p = ⟨path⟩ ; comb(p,300); draw p;`

using the `comb` macro that will be presented here (the `300` scales the curvature comb).

We start by defining the macro `crossprod` that returns the cross product between two given vectors $\vec{w}$ and $\vec{z}$:

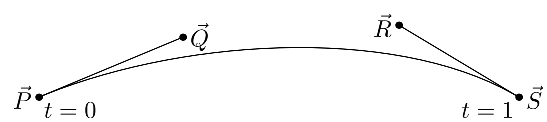
```
primarydef w crossprod z =
 (xpart w * ypart z - ypart w * xpart z)
enddef;
```

Then the macro `curv` will be applied to a path `p`, returning a "curvature vector" that is orthogonal to the path in its initial point. The length of the vector is proportional to the initial curvature of the path:

```
vardef curv expr p =
 save v,w,l; pair v,w;
 v = direction 0 of p;
 l = length v;
 v := v/l;
 w = (point 0 of p - 2*postcontrol 0 of p
                    + precontrol 1 of p)/l;
 2/3*(v crossprod w)/l*(v rotated -90)
enddef;
```

Here is the math behind this macro. A cubic Bézier segment can be described by:

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = t^3(3\vec{Q} - \vec{P} + \vec{S} - 3\vec{R}) + 3t^2(\vec{P} - 2\vec{Q} + \vec{R}) \\ + 3t(\vec{Q} - \vec{P}) + \vec{P}$$



The initial derivatives are then:

$$\begin{pmatrix} \dot{x}(0) \\ \dot{y}(0) \end{pmatrix} = 3\underbrace{(\vec{Q} - \vec{P})}_{=:\vec{v}} \qquad \begin{pmatrix} \ddot{x}(0) \\ \ddot{y}(0) \end{pmatrix} = 6\underbrace{(\vec{P} - 2\vec{Q} + \vec{R})}_{=:\vec{w}}$$

The signed curvature is calculated by $\dfrac{\binom{\dot{x}}{\dot{y}} \times \binom{\ddot{x}}{\ddot{y}}}{\left|\binom{\dot{x}}{\dot{y}}\right|^3}$.

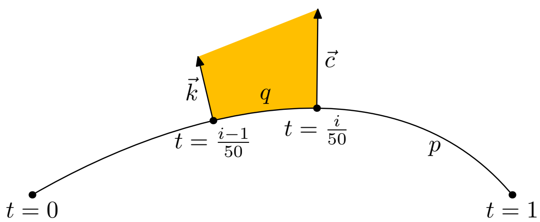
Using $l := |\vec{v}|$ we finally have the formula used in the macro for the initial curvature:

$$\frac{3\vec{v} \times 6\vec{w}}{(3l)^3} = \frac{2}{3}\frac{\vec{v} \times \vec{w}}{l^3} = \frac{2}{3l} \cdot \left(\frac{1}{l}\vec{v} \times \frac{1}{l}\vec{w}\right)$$

The separated divisions by l are necessary to prevent arithmetic overflows. The special case $\left|\binom{\dot{x}(0)}{\dot{y}(0)}\right| = 0$ is not handled here. The curvature then would diverge to $\pm\infty$ (or be 0 if the cubic Bézier segment is a line segment).

Now we define the curvature comb macro of a path p by subdividing each segment in 50 parts and filling an area over each part. Each part of the comb is made of two subsequent "curvature" vectors $\vec{k}$, $\vec{c}$ that are scaled by a constant factor s given by the user. The color depends on their average length.

doi.org/10.47397/tb/44-2/tb137romer-curvetools




```
vardef comb(expr p,s) =
 save q,c,k; path q; pair c,k;
 for n = 0 upto length(p)-1:
  c := s * curv subpath(n,n+1) of p;
  for i = 1 upto 50:
   k := c;
   c := s * curv subpath(n+i/50,
             n if i<25: +1 fi) of p;
   q := subpath(n+(i-1)/50,n+i/50) of p;
   fill q -- point 1 of q + c
          -- point 0 of q + k
          -- cycle withcolor
      (1,1/(1+.1*.5[length c,length k]),0);
  endfor
 endfor
enddef;
```

The condition `if i<25: +1 fi` makes the subpath as large as possible to get better accuracy.

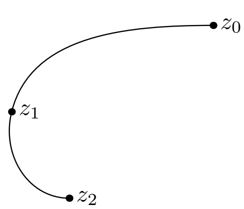
A curvature of 0 is mapped to yellow and an infinitely large curvature is mapped to red. This is done by changing the green value between 1 and 0. If the `.1` is increased, the green value tends faster to 0.

3 Harmonize paths in METAPOST

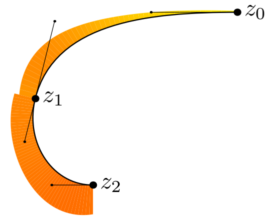
In METAPOST, the code

```
z0 = (70,60); z1 = (0,30); z2 = (20,0);
draw z0{left} .. z1 .. z2{right};
```

produces the following curve:

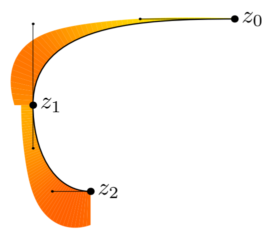


While the directions at the start and the end of the path were set by the user, METAPOST has chosen the angle of the path in z_1 to equalize the so-called *mock curvature* on both sides. The mock curvature is a Taylor approximation of the real curvature [1]. After that, two cubic Bézier segments that nearly minimize the curve energy have been drawn between the given points.



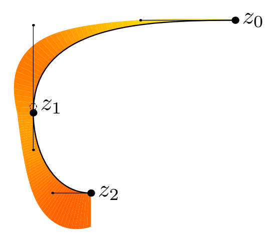

```
   z0{left}
.. z1
.. z2{right}
```

The curvature comb in the preceding picture shows that the curvature at z_1 indeed is not continuous but only near-continuous. When a user sets the direction in the joining knot, METAPOST has no possibility of optimizing the curvature in the joint and the curvature often changes more abruptly at the joint (see the following picture). This case is frequent in type design because knots at horizontal and vertical extrema are preferred over knots with arbitrary direction.




```
   z0{left}
.. z1{down}
.. z2{right}
```

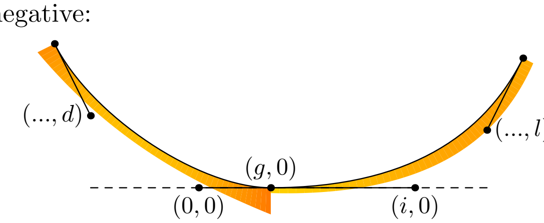
Fortunately, a METAPOST path can be modified to a continuous-curvature curve with the `harmonize` macro we'll define later. It moves the joining knot along its tangent:




```
harmonize z0{left}
       .. z1{down}
       .. z2{right}
```

4 The math of harmonization

Assume two adjoint cubic Bézier segments that have the same direction at their join and do not have zero-handles. Furthermore, assume the joining knot is not a point of inflection. By translation and rotation we can force one control point next to the joining knot to lie on the origin of the coordinate system and the joining knot tangent to lie on the x-axis. Additionally, the depicted coordinate d will be non-negative:

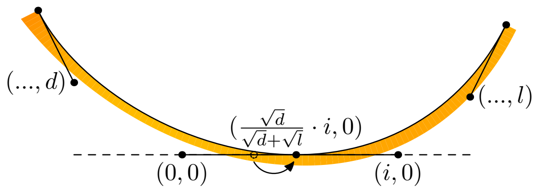


We want to choose g such that the curvature is continuous. So the curvature on both sides of $(g, 0)$ must be equal:

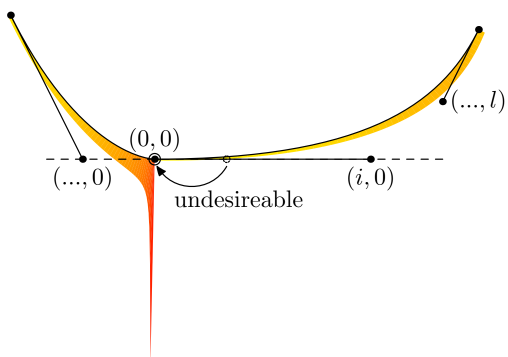
$$\frac{2d}{3g^2} = \frac{2l}{3(i-g)^2}$$

Solving for g and obeying the condition $0 \leq g \leq i$ we get

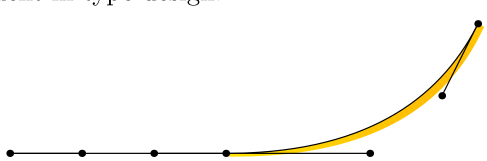
$$g = \frac{\sqrt{d}}{\sqrt{d} + \sqrt{l}} \cdot i.$$



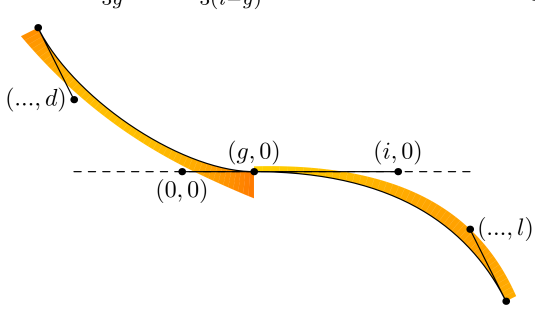
If either d or l is zero, $g = \frac{\sqrt{d}}{\sqrt{d}+\sqrt{l}} \cdot i$ becomes either 0 or i. That means the joining knot will become collocated with one of its control points, which generally should be avoided. One reason for this avoidance is that the curvature might become infinitely large:



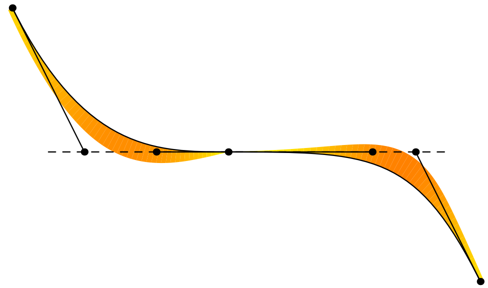
So, we will not alter the paths at all in the case of either d or l being zero. This case occurs also when a straight line goes over to a curve, which is quite frequent in type design:



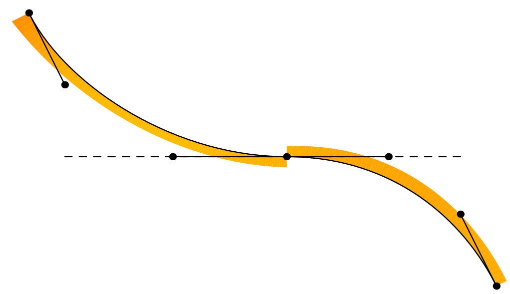
When the joining knot is a point of inflection, the curvatures $\frac{2d}{3g^2}$ and $\frac{2l}{3(i-g)^2}$ must have different signs.



Hence, a curvature-continuous solution forces $d = l = 0$, so all control points must lie on one line, as in:



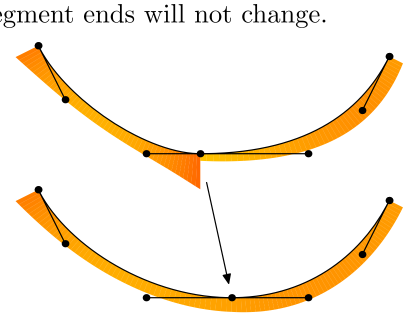
In this situation, one could also satisfy further conditions like the preservation of area. On the other hand, having all four control points on the same line as the two affected cubic Bézier segments is critical. Due to rounding errors, such a conversion may add new points of inflection. So, instead of this, we will only guarantee the *absolute value* of the curvature to be continuous in the case of points of inflection by moving the joining knot in the same manner as before, between its control points:



Finally, the solution of setting

$$g_{\text{new}} = \begin{cases} g_{\text{old}} & \text{if } d = 0 \text{ or } l = 0, \\ \frac{\sqrt{|d|}}{\sqrt{|d|}+\sqrt{|l|}} \cdot i & \text{else} \end{cases}$$

is chosen here and shall be the definition of *harmonization*. By harmonization the curvatures at the other segment ends will not change.



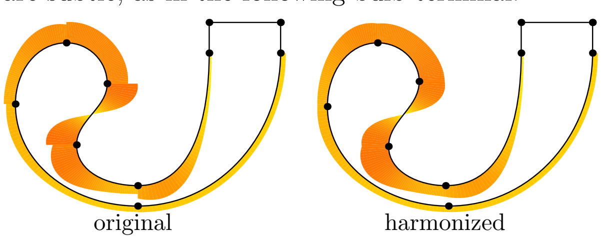
We define a corresponding macro `harmonize` that returns a harmonized version of a given path p. In the generic case, the tangent in the joining knot is not the x-axis (as depicted in the preceding figures), so we calculate d and l as the height to the tangent by cross products.

```
vardef harmonize expr p =
 save t,u,d,l,n,q; pair t,u,q[];
 n = length p;
 for j = if cycle p: 0 else: 1 fi upto n-1:
  q[j] = point j of p;
  t := unitvector(direction j of p);
  u := unitvector(point j of p
          - precontrol j of p);
  if eps > abs((u dotprod t) - 1): % smooth
   l := abs(t crossprod
       (precontrol j+1 of p - point j of p) );
   d := abs(t crossprod
       (postcontrol j-1 of p - point j of p) );
   if not ( (l = 0) or (d = 0) ):
    q[j] := if (d = l): .5 else:
        ( sqrt(d) / (sqrt(d) + sqrt(l)) ) fi
        [precontrol j of p,postcontrol j of p];
   fi
  fi
 endfor
 if not cycle p:
  q[0] = point 0 of p;
  q[n] = point n of p;
 fi
 q[0] % start returned path
 for j = 0 upto n-1: % define new path
  .. controls postcontrol j of p
  and precontrol j+1 of p .. if (j = n-1)
  and (cycle p): cycle else: q[j+1] fi
 endfor
enddef;
```

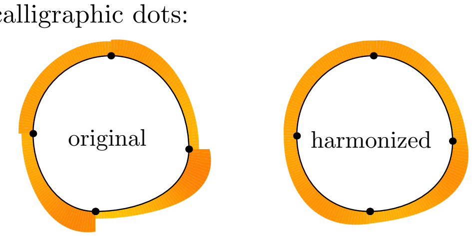
A mostly equivalent algorithm has been published in [2].

5 Examples of harmonization

Should you use harmonization? At least it does no harm to consider it. Most of the time, the changes are subtle, as in the following bulb terminal:



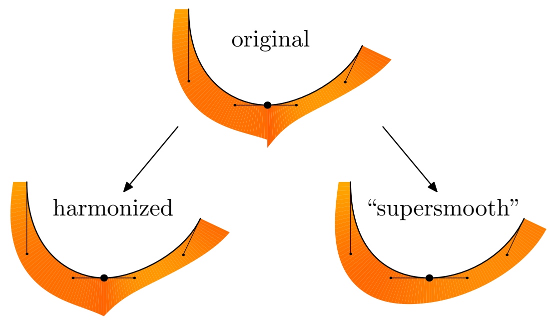
And sometimes they are less subtle, as in the following calligraphic dots:



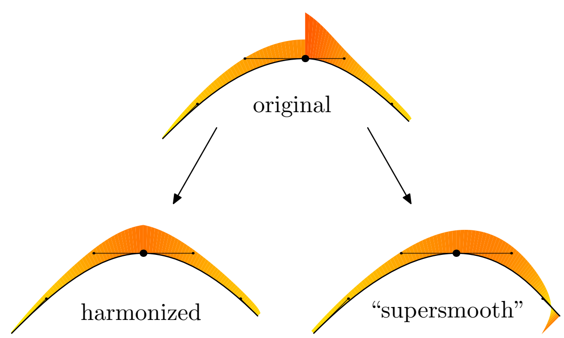
After harmonization, the dot has become rather more rounded and may have lost its "personality".

6 Smoothing out paths even more

Since harmonization does not affect off-curve control points nor the curvatures at other joining knots it can be easily used over several cubic Bézier segments. Nonetheless, harmonized paths normally no longer interpolate the knots they were originally meant to. The author once thought it might therefore be a good idea to leave the joining knots and move the control points instead. Then we not only can make the curvature continuous but also the change of curvature. The curve then becomes something that may be termed "supersmooth".



However, there are some problems that come with this "supersmoothness": It might introduce additional points of inflection (see below). Furthermore, this will normally change the curvature at other knots and break curvature continuity there.



References

[1] J.D. Hobby. Smooth, easy to compute interpolating splines. *Discrete & Computational Geometry* 1(2):123–140, 1986. doi.org/10.1007/BF02187690

[2] R.L. Roach. Curvature continuity of cubic Bezier curves in the solid modeling aerospace research tools design software. Interim report, NASA Langley Research Center, 1990. ntrs.nasa.gov/citations/19900012238

◇ Linus Romer
Kantonsschule Glarus
Winkelstrasse 1
8750 Glarus (Switzerland)

Using Asymptote like METAPOST

Jim Hefferon

Abstract

Asymptote is a vector graphics language for technical drawing that fits very well with TeX, LaTeX, and friends. It deserves to be more widely known.

One appealing thing is that it is in part based on algorithms from METAFONT and METAPOST but it extends those to three dimensions. I'll discuss a couple of workflow issues that a beginner to Asymptote who is coming from METAPOST might find useful, in particular using a single source file to output many related graphics.

1 Introduction

I wrote a book years ago using METAPOST and found that it had many advantages. I cannot draw, at all, and it was a comfort to be able to tell the computer to, say, make this line to be exactly two thirds the length of that other line.

But the best feature is that METAPOST fits a person who thinks mathematically. For example there is a simple way to find where two lines intersect.

However, having worked with METAPOST a lot, I was aware of some warts. For me the two biggest are lack of any real 3D abilities, and that programming in the language can be ... quirky. So when I saw the new Asymptote system I was eager to try it. It has been very good.

2 Overview

Asymptote is a powerful descriptive vector graphics language. It provides a natural coordinate-based framework for technical drawing. LaTeX typesets the text and equations.[1]

- It is inspired by METAPOST, including generalizing METAPOST's path construction algorithms to three dimensions.[2] It inherits METAPOST's impedance match with a mathematical mindset. But it is a more standard programming language, including declared types, familiar syntax, and IEEE floating point numbers.
- It outputs high-quality PostScript, OpenGL, PDF, SVG, WebGL, and V3D, as well as 3D vector WebGL graphics for HTML files and 3D vector PRC graphics for PDF files.
- It uses deferred drawing to solve size constraint issues between fixed-sized objects, such as labels and arrowheads, and objects that should scale with figure size.
- It runs on Unix, macOS, and Windows, and is under continuing development.

You can even try it in your browser without installing it, using the Asymptote Web Application.[3]

3 Compared to TikZ

Many readers will be familiar with TikZ so I will briefly compare with that system.

I have not used TikZ much, and have not used it lately at all. But I lined up the two when I was starting my latest project, some years ago. They have many of the same strengths. I found that Asymptote had some technical advantages, including the native 3D graphics, and I also found TikZ harder to program in.

Another difference, relevant here, is that the basic paradigm in TikZ is that your figures are in your document, generated when the document is generated. In Asymptote the paradigm is that they are generated outside the document. (Yes, you can generate stand-alone in TikZ and yes, you can include Asymptote code in a document.)

I had a bad experience in the past when I was using PSTricks and the LaTeX world switched to pdfLaTeX. That left me with a preference for a dependence chain that is shorter and broader over one that is taller and thinner. I saw this again recently when there was an issue with Ghostscript and I was able to put the external PDF figures into my repository until the issue was resolved. So one factor in my comparison tipping me to Asymptote was a preference for generating figures independently from the documents.

4 Working in METAPOST

The basic structure of METAPOST input is that one file holds many figures. This will output a graphic into a file numbered 1 and another graphic into a file numbered 2.

```
beginfig (1)  % MetaPost
  % first figure drawing commands
endfig;
beginfig (2)
  % second figure drawing commands
endfig;
```

Putting a number of related sources in the same file is convenient. For instance, if these are drawings for a calculus lecture then you might at the top of the file declare `VECTOR_THICKNESS=0.8pt`, and use that in lots of the figures.

[1] asymptote.sourceforge.io/asymptote.pdf

[2] asymptote.sourceforge.io/gallery/3Dgraphs/

[3] http://asymptote.ualberta.ca/

doi.org/10.47397/tb/44-2/tb137hefferon-asymptote

I will describe two adjustments that may help a person coming from METAPOST and that took me some time to dope out.

5 Adjustment one: multiple figures per file

Here is the skeleton to put multiple figures in one Asymptote file.

This outputs a file test000.pdf containing the graphic with a diagonal red line.

```
string OUTPUT_FN = "test%03d";
// =============
picture pic; int picnum = 0;
unitsize(pic, 1cm);  // dist from x=0 to x=1
draw(pic, (0,0)--(1,1), red); // make a line
shipout(format(OUTPUT_FN,picnum),
         pic, format="pdf");

// =============
picture pic; int picnum = 1;
  ...
shipout(...);
```

Some things to notice in that code:

- The OUTPUT_FN gives all file names the same structure. So if you have lots of graphics (my current book has more than 2000) then it is easier to work with them. Of course, %03d gives the picture number as a three decimal place integer. I find two decimal places is too tight.
- A second picture begins with the declarations picture pic and int picnum = 1 and ends with an identical shipout(...), as shown.
- I write picnum = 0 and picnum = 1, etc., instead of picnum = picnum+1. When you go back a week later into a file with eighty pictures, looking to fix a bug in the fifty-third, you want it this way.
- Because of the multiple outputs from a single input, lots of commands need a picture argument. The code has it in the draw(...) command. If you leave out the pic then the line gets drawn somewhere (that is, there is no error) but not in the output file where you are looking for it. The pic is also in the unitsize(...) command.

A comment: METAPOST is set up so the figure code blocks are isolated, in that if, say, you set x=1 inside one beginfig ... endfig then that does not affect an x inside another. There is no such isolation here. But I sometimes reuse variables in a number of related figures so I'm happy with this.

6 Adjustment two: style files

This is related to the prior adjustment in that one way that having multiple outputs from a single input is helpful is to enforce uniformity. You can have parameters such as line thickness or font size, and specify them in just the one file.

But the same applies across multiple files. It is useful to have every Asymptote source file get desired global parameters from a single file.

You bring in a file with the import fn command. For instance, Asymptote has a standard file called settings.asy that you usually want to bring in.

```
import settings;
settings.outformat="pdf";
```

If that file is in the list of directories searched by the Asymptote system then you are good. The list is what you might guess: first the current directory, then one or more directories given by the environment variable ASYMPTOTE_DIR, etc.

Here is part of the style file for my current book.

```
import fontsize;
defaultpen(fontsize(9.24994pt));
import texcolors;
pen darkgrey_color=rgb("595241");
pen lightgrey_color=rgb("E0D4BE");
pen white_color=rgb("FFFFFF");
pen lightblue_color=rgb("ACCFCC");
pen red_color=rgb("8A0917");
// Use these names, not prior ones
pen highlightcolor=red_color;
pen backgroundcolor=lightblue_color;
pen boldcolor=darkgrey_color;
pen lightcolor=lightgrey_color;
pen verylightcolor=white_color;
```

(I like the color names such as highlightcolor to make graphics because I sometimes adjust the color scheme, for instance when I get a proof copy from the publisher.)

In my current project I also have a LaTeX style file containing the font information and document-specific macros that is used by every Asymptote file, as well as by the LaTeX driver file book.tex, ensuring that the figures match the text.

7 Ending

Asymptote does a great job drawing technical graphics. Adding some METAPOST-like workflow makes it even more fun.

◇ Jim Hefferon
Mathematics and Statistics
University of Vermont
jim.hefferon (at) gmail dot com
https://hefferon.net

Interactive and real-time typesetting for demonstration and experimentation: ETAP

Didier Verna

Abstract

We present ETAP, a platform designed to support both demonstration of, and experimentation with digital typesetting, interactively, and in real-time.

ETAP provides a GUI which currently focuses on all aspects involved in paragraph formatting. A number of pre-processing features can be switched on or off (hyphenation, kerning, ligaturing, *etc.*). A specific paragraph formatting scheme may be selected from a pool of ready-made algorithms, and adding new algorithms to that pool is easy. Each algorithm comes with its own set of configuration parameters, and the GUI allows you to tweak those parameters and observe the effects in real-time.

ETAP may also be used without, or in parallel with the GUI. While the application is running, the whole programmatic infrastructure is manipulable from a command-line interface. This allows inspection of the various typesetting objects normally displayed by the GUI, and also to perform computations with them, for example, data collection and statistical measurements.

1 Introduction

The world of digital typesetting is a fascinating one. As an application domain, it combines a strong focus on aesthetics with many interesting technical challenges, thus making it an Art as much as a Science. The motivation for the project described in this paper is twofold: experimentation (the Science) and demonstration (the Art).

Experimentation Suppose you want to try out a new ideas for paragraph justification. Experimentation (including rapid prototyping and debugging) would be made a lot easier with a direct visualization of the results on a sample text (actual contents not necessarily important), and with the ability to interactively tweak such or such parameter from a GUI (Graphical User Interface), while observing the effects in real time.

Demonstration In terms of demonstration, my personal experience (in particular when trying to raise students' awareness of the beauty and the subtlety of high quality typesetting) is that showing off static pages of text simply doesn't cut it. On the other hand, there is nothing like having the ability to switch kerning on and off, and immediately see the result, to strike people's minds. The same goes for ligaturing, with no characters actually displayed, but only their bounding boxes which, all of a sudden, go from two or three to just one.

By now, the reader has noticed that whether it is for experimentation or demonstration purposes, the system(s) we are talking about share two common traits: they need to be interactive, and work in real-time. It turns out that, if given those two properties, there is no reason why a single such system couldn't fulfill both objectives. The purpose of this paper is precisely to exhibit one such possible system.

In general, typesetting experimentation is not a very practical thing to do. WYSIWYG (What You See is What You Get) systems are very reactive (for example, you can see the paragraphs being formatted as you type them) but provide neither the highest rendering quality, nor the highest degree of configurability, let alone extensibility. TeX [8, 9], on the other hand, is renowned for the quality of its rendering, but works more like a non-interactive programming language, with its separate development / compilation / visualization phases.

Granted, there are several attempts at bridging the gap. Overleaf, BaKoMa, LyX, and TeXworks provide WYSIWYG environments to TeX, increasing the interactive "feel". Batch Commander [3] was an attempt at providing a GUI for TeX configuration. LuaTeX provides some level of access to TeX's internals. However, none of these systems would let you fundamentally change the way TeX works (they are not meant to).

We must also mention TeXmacs, a very interesting project not in fact based on TeX, but still providing high-quality typesetting from within a WYSIWYG environment. TeXmacs is written in C++ and embeds a Guile interpreter (a dialect of Scheme, from the Lisp family) as an extension language. This makes the project very close to LuaTeX, at least in spirit. This also makes it share the same characteristics: it is a heterogeneous platform, and the C++ core is neither interactive, nor easily modifiable.

As a matter of fact, most available systems today would fail the experimentation goal for a simple reason: they are *production* systems.

We present ETAP (Experimental[1] Typesetting Algorithms Platform), a tool written to ease typesetting experimentation and demonstration. ETAP currently focuses on paragraph formatting, and provides an extensible list of configurable algorithms. ETAP

[1] Whether the "experimental" part refers to typesetting, algorithms, platform, or a combination of them is left to the discretion of the user. . .

doi.org/10.47397/tb/44-2/tb137verna-realtime

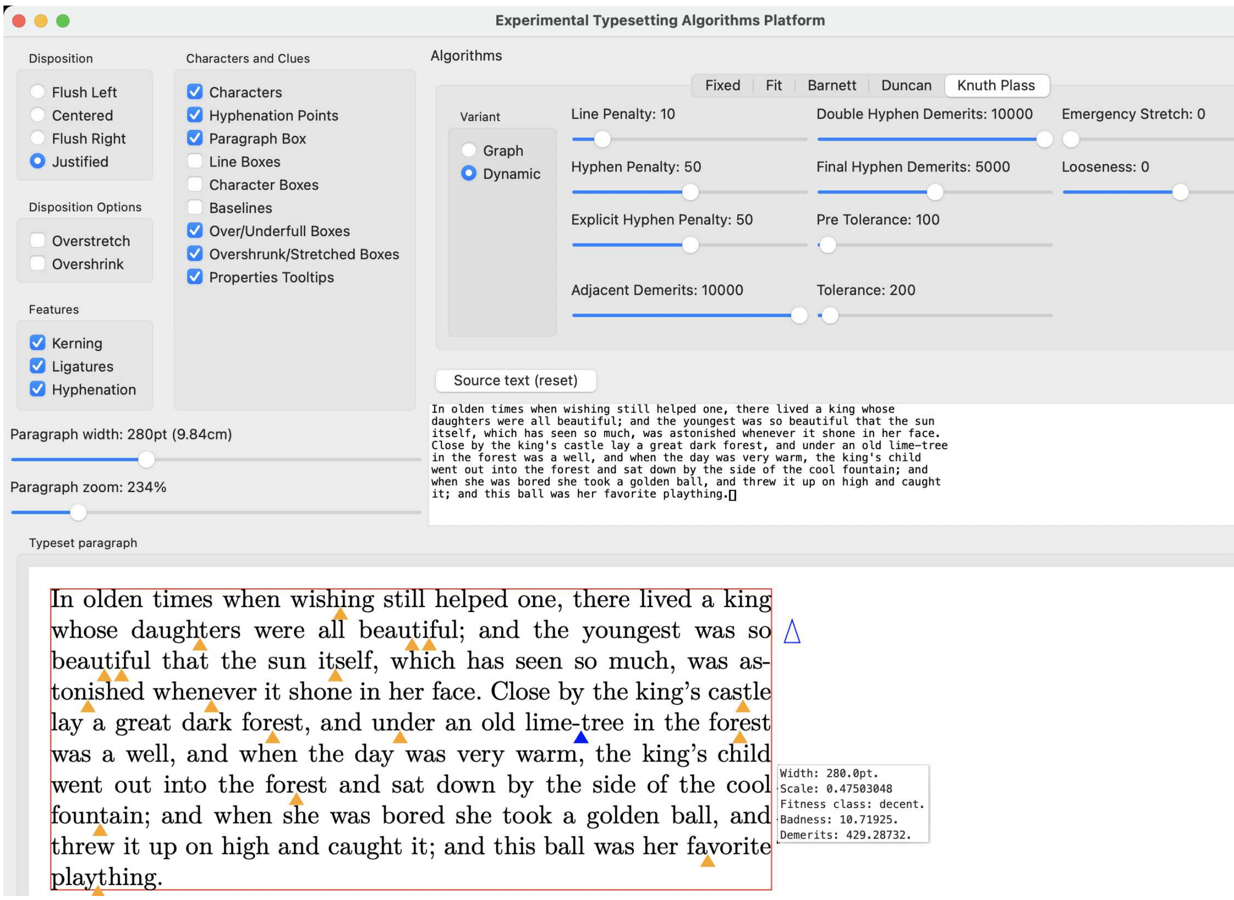


Figure 1: The ETAP GUI

also features switchable kerning, ligaturing, and hyphenation. The source text is editable, and the resulting paragraph is (re)displayed in real-time, along with many switchable visual hints, such as paragraph, character, and line boxes, baselines, over/underfull boxes, hyphenation clues, *etc.* All these parameters, along with the desired paragraph width, are adjustable interactively through the GUI.

But ETAP can also be used without, or in parallel with, the GUI, as a scriptable application. This comes directly from its homogeneous design: the application is written entirely in one industrial-strength programming language, Common Lisp [1], which is multi-paradigm, dynamic, and interactive. In particular, fully reflexive access to the various internal data structures, and in fact, to the whole program while it is running, allows for considerable experimentation opportunities, such as batch-formatting in various environmental conditions, and data collection for empirical evaluation and statistical measurements.

Section 2 provides a description of the application's most important features, focusing on interactive manipulation through the GUI. Section 3 discusses some software engineering aspects of its implementation. Finally, Section 4 describes the programmatic (interactive, yet non-graphical) capabilities of ETAP for experimentation.

2 The platform

Note: for the interested reader, another, slightly different description of the platform is available in [16].

A screenshot of ETAP's GUI is provided in Figure 1. For as much as an interactive and real-time application can be described on paper, the picture should at least give the reader a general feeling of what is available.

The Knuth-Plass algorithm [10] has been selected, along with the default values for all its parameters. The source text for the paragraph is typeset accordingly, in justified disposition, and with kerning, ligatures (although there are none here), and hyphenation.

A number of visual clues have been activated as well and can be observed in the paragraph rendering area (the bottom half of the window). In addition to the characters themselves, the paragraph's bounding box is drawn. The small arrows pointing upward between characters represent the hyphenation points at which the algorithm has decided *not* to break lines. Finally, the unfilled triangle to the right of line 2 indicates an intentionally overstretched line. This means that the algorithm has decided on a scaling (of glue) ratio which exceeds 1. Indeed, the Knuth-Plass algorithm ran twice here, using a tolerance threshold of 200 the second time. One can also observe that the third line had to be hyphenated, which confirms this is not the result of pass 1 of the algorithm.

Finally, one can see a small popup window near the bottom-right corner of the typeset paragraph. This is actually a "properties tooltip" which pops up when the mouse is moved over a line, and provides feedback on the line in question. In this particular case, it indicates that the line is 280pt wide (the paragraph's width, as the line is properly justified), and is stretched by a scaling factor of approximately 0.475. Also, because the selected algorithm is the Knuth-Plass one, the tooltip reports the line's fitness class, badness, and local demerits. If we were to move the mouse over the paragraph's left margin, the tooltips would advertise a number of global paragraph properties, such as the total demerits, the algorithm's pass number, and the number of remaining active nodes at the end of execution.

Since we are talking about the Knuth-Plass algorithm, note that this project does not aim at providing an exact replica of it, nor of any other currently available line-breaking algorithms (notably Barnett [2] and Duncan [6]), nor of any future ones. In fact, it is our opinion that what is called the "Knuth-Plass algorithm" is actually *not* an algorithm *per se*, but rather the combination of a typical shortest path finding algorithm with a particular cost function having the suitable properties for dynamic programming optimization, all of this written in a relatively low-level imperative language with performance concerns of that time (the 1980s) in mind.

On the other hand, what we are interested in is providing an exact replica of the algorithm's *logic*. Common Lisp is a much higher-level programming language, and most performance concerns of the time have long been obsoleted by the continuously increasing computing power at hand (besides, performance is rarely a top priority for an experimentation platform). Consequently, our design and choice of precise data structures diverge from the original. For example, we actually provide two different implementations of the Knuth-Plass algorithm: one, close to the original, equipped with the same dynamic programming optimization, and another one based on the exploration of a complete graph of solutions (*not* the brute force and exhaustive 2^n one, though!).

Another example where we differ from the original is, again, motivated by demonstration and experimentation. In the original Knuth-Plass, pass 1 of the algorithm works on a non-hyphenated text (hyphenation was considered too costly at the time). Only if that fails does TeX hyphenate the text and try a second pass (also with a different tolerance threshold). In our case, we want to be able to display the hyphenation clues every time, if so requested. Consequently, the hyphenation process is implemented as a global option (independent of the selected typesetting algorithm), and pass 1 of the Knuth-Plass algorithm may consequently run on an already hyphenated text, in which case it simply disregards the hyphenation points as potential break points.

3 Software engineering

In the context where TeX is still one of the best typesetting systems out there, but also one of the oldest, we deem it important to say a word about software engineering. It is a well-known fact that the science of programming languages and paradigms has evolved considerably over the years. Some people have written about the virtues of a purely functional approach to paragraph breaking in the past [4, 13]. We, on the other hand, favor a more pragmatic than theoretical approach. In particular, instead of having a single paradigm (*e.g.*, functional programming) imposed on us, we prefer the freedom and flexibility provided by a multi-paradigm language [15].

Virtually any programming paradigm aims at increasing both the code's clarity and concision at the same time. Table 1 provides a rough estimate of the project's size in LoC (Lines of Code), and clearly illustrates the benefits of being multi-paradigm for concision. Liang's hyphenation algorithm [11] amounts to 150 LoC. The 500 lines of "lineup" correspond to the pre-processing of the source text, including hyphenation, kerning, ligaturing, and glueing. The currently available paragraph formatting algorithms comprise between 150 and 450 LoC (each variant of Knuth-Plass takes 350 lines). We believe this

Table 1: Rough estimate of ETAP's size

	LoC
GUI	800
Hyphenation	150
Lineup	500
Algorithms	150–450
Knuth-Plass	350 per variant

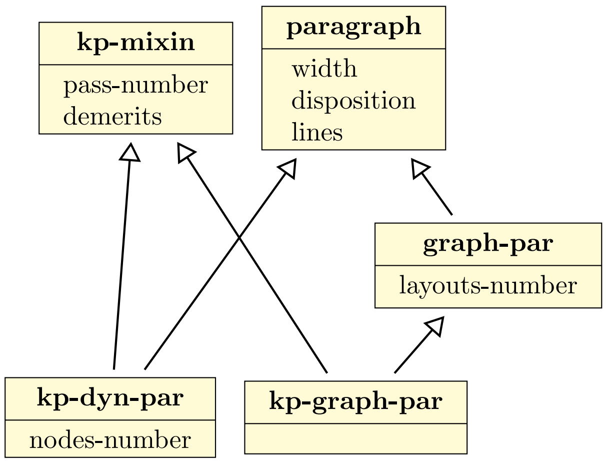


Figure 2: The `paragraph` classes

makes the whole platform rather small, considering the functionality offered. In fact, the whole thing currently remains below 5000 LoC, exclusive of large data blocks: an additional 5000 lines is the "lispification" of Adobe's glyph list, and another 5000 lines for English hyphenation patterns.

Let us now illustrate why being multi-paradigm is beneficial for the project, via some examples.

3.1 Object orientation

Traditional, class-based object orientation revolves around two fundamental concepts: inheritance (organization of the data) for code reuse, and polymorphism (manipulation of the data) for genericity.

Figure 2 depicts the `paragraph` class hierarchy in a UML fashion. Every subclass incorporates the contents of its superclass(es), thus avoiding duplication. The presence of multiple inheritance (not available in all object-oriented languages) ensures maximum sharing of code: the `kp-dyn-par` class represents paragraphs typeset with the dynamic programming variant of the Knuth-Plass algorithm. Such a paragraph *is* a regular paragraph before anything else, but it also is a `kp-mixin` one, which means that it remembers its pass number and total demerits. Finally, this class also has one additional property of its own: the number of remaining active nodes when the algorithm terminates.

The GUI is passed a paragraph object which, most of the time, is an instance of one of the subclasses (for example, a `kp-dyn-par`). But the visual rendering function is interested only in the contents of the base class (it needs to know only the paragraph's width, disposition and lines to perform the formatting) so it is in fact unaware of the object's exact class. On the other hand, the properties tooltip popup is implemented via a polymorphic generic function with different implementations for every paragraph class. That is why, in a single function call, it can still advertise the `nodes-number` properties when available, and simply doesn't otherwise.

```
;; Duncan
(make-graph lineup width)

;; Knuth-Plass, graph variant
(make-graph lineup width
  ;; alternative "next boundaries" function...
  :next-boundaries #'kp-next-boundaries
  ;; ... plus some specific arguments.
  :threshold pre-tolerance)
```

Figure 3: The `make-graph` higher order function

3.2 First class functions

The second example is that of functional programming, although not in the "purely functional" sense mentioned earlier, but rather in Christopher Strachey's sense [5, 14]. Functions in a programming language are said to be "first class", or "first order", or even "higher order" if they behave like any other kind of object: they can be created dynamically, passed as arguments to other functions, provided as return values, *etc.*

Figure 3 provides an illustration of how functional programming contributes to concision as much as object orientation, only in a different way. ETAP has a function called `make-graph` which accepts a lineup and a paragraph width as arguments, and returns a graph of all possible break point solutions. By default, starting at a specific position in the lineup, the next possible break points would be those involving a scaling of at most 1 in absolute value. There is a function called `next-boundaries` which computes the list of such break points.

On the other hand, some algorithms may have a different view on what the next possible break points actually are. For instance, the Knuth-Plass algorithm does not look at the scaling alone, but considers hyphenation and uses a pre-tolerance or a tolerance threshold, depending on the pass number. Creating a Knuth-Plass graph thus only differs from a regular one in the way the next possible break points are computed. It would be unsatisfactory to write a specific version of `make-graph` just because of that small divergence from the default behavior. In fact, most of the code would actually be redundant with the regular version.

What we do instead, is parameterize the "next boundaries" function. The Knuth-Plass implementation comes with an alternative function called `kp-next-boundaries`. As you can see in Figure 3, make-graph is in fact a higher order function, accepting a “next boundaries” function as argument. This ensures that the skeleton of make-graph does not need to be duplicated.

4 Experimentation

The last critical software engineering aspect which we want to emphasize is the dynamic and interactive nature of Common Lisp, ETAP’s implementation language. Just like the more mainstream scripting languages such as Ruby, Python, or Perl, Lisp provides a REPL (Read Eval Print Loop) from which the programmer can interact with the program while the program is running. In fact, both the REPL and the GUI may be used at the same time to interact with the system, and the homoiconic [7, 12] nature of the language makes it trivial to introspect the live objects, or even destructively modify them.

A typical experimentation scenario is as follows. The programmer runs a typesetting experiment via the GUI in various conditions, and observes a surprising (or suspicious) situation. The programmer then switches to the REPL and from there, has the ability to inspect (or debug) the complete program state, without leaving the program. It is even possible to hot-modify the typesetting code, for example to fix a bug and switch back to the GUI in order to trigger a redisplay.

Let us now illustrate the benefits of interactivity with two examples, the second being a recent and true anecdote.

4.1 Statistics

ETAP provides a short (around 200 LoC) generic infrastructure for data collection and statistical measurements of all sorts. For example, there is a function called scalar-statistics that loops over all available algorithms and paragraph widths, and each time collects a scalar value computed by a function passed as an argument (another case of functional programming at work).

This function can be used to generate comparative charts for any criterion one may think of. For example, with the two function calls below, we are able to generate the charts presented in Figures 4 and 5.

```
(scalar-statistics #'collect-scales-mean)
(scalar-statistics #'collect-scales-variance)
```

Those charts are primarily meant to be visualized on (large) screens, so they will appear somewhat cluttered in a PDF. Their actual content is not so important here, as the point is merely to illustrate the current capabilities of the platform.

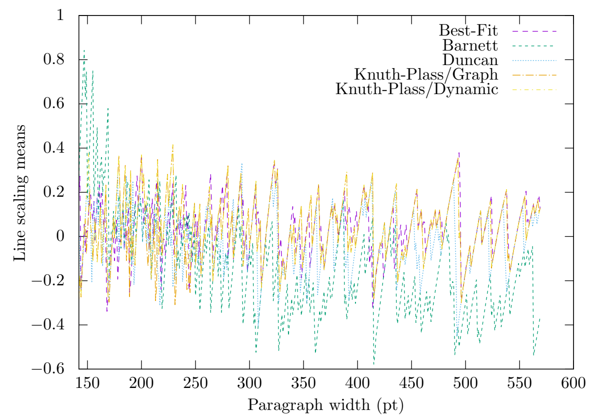


Figure 4: Scales mean

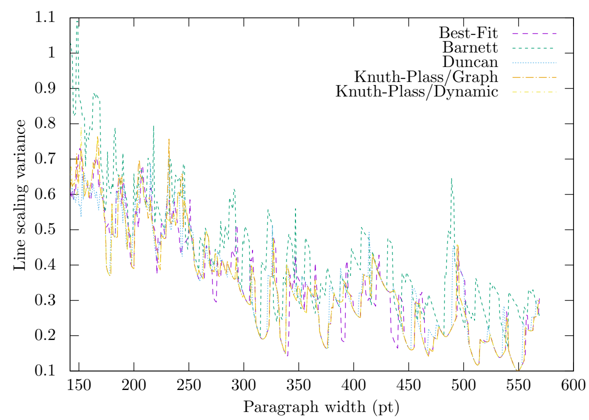


Figure 5: Scales variance

The first one shows the average line scaling for every algorithm and paragraph width. It is immediately visible on this chart that except for very narrow paragraphs, the Barnett algorithm has a tendency to compress a lot. On the other hand, the second chart also shows that Barnett has a higher scaling variance than the other algorithms. In TEX’s terms, this means that the adjacent demerits would probably be off the charts (and that would be easy to confirm too).

These two charts are just examples. Other ready-made data collection functions allow you to compute the graph sizes, the number of possible solutions, the number of under/overfull lines, *etc.*, usually in less than 15 LoC.

4.2 The anecdote

The final example we want to provide here takes the form of an anecdote, and we think it illustrates pretty well why having such a platform for experimentation is convenient.

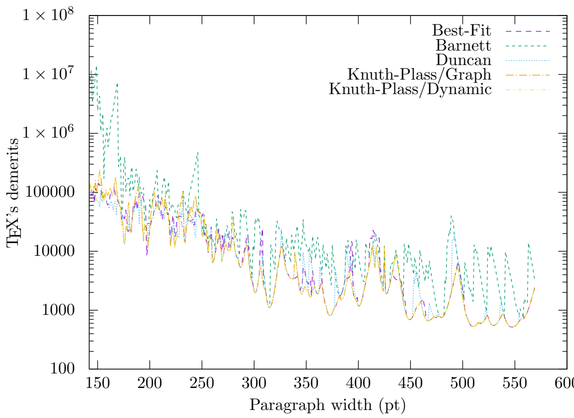


Figure 6: TEX's view of the competition

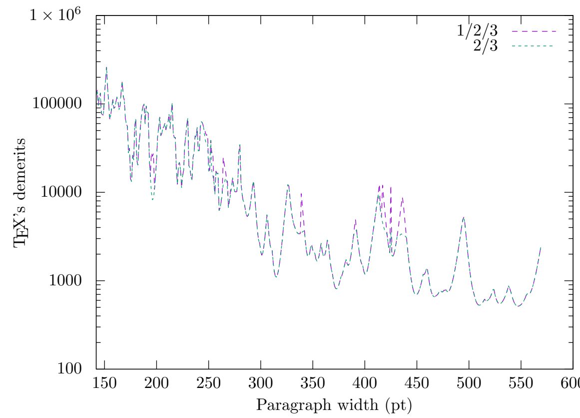


Figure 7: With or without pre-tolerance

At some point, we were curious to get an idea of TEX's view of the competition. In other words, the question was: given a paragraph formatted by an algorithm other than Knuth-Plass, what is this paragraph's number of total demerits?

We thus wrote a 50-line function to compute the chart presented in Figure 6. Of course, the expected general result is that when there is a valid paragraph breaking solution, TEX should find itself better than the competition, because again, it *does* find optimal solution according to its own quality criteria. Figure 6 does confirm this, although if you look closely, you will spot a curious area, near the 440pt paragraph width, where the Best Fit algorithm seems to perform better.

At first, we thought there was a bug somewhere in the implementation of one algorithm or the other, but in fact the code was correct. Visualizing the resulting paragraphs made it apparent that the Best Fit solution contained hyphens, and the Knuth-Plass one didn't. Recall that TEX will stop at pass 1 of the algorithm if it finds a valid solution without hyphens.

The next hypothesis, then, was that there would be cases in which a hyphenated paragraph would amount to fewer demerits than its non-hyphenated counterpart. Once you come to think of it, this hypothesis is in fact very plausible, given that by default, hyphen penalties are quite small (50) compared to, say, adjacency penalties (10000).

We were able to confirm that hypothesis in literally one line of code. Indeed, generating the chart presented in Figure 7 took only one function call. This chart plots the demerits for all paragraph widths with or without pass 1 of the Knuth-Plass algorithm. Recall that short-circuiting pass 1 is done by setting the pre-tolerance parameter to -1 (which is what LATEX does, by the way).

On this chart, some areas are clearly visible where pass 2 of the algorithm performs better (in terms of total demerits) than pass 1. It is a bit surprising that after 25 years of using LATEX, we only recently realized that. But the important point, here is how ETAP made the experimentation, hypothesis formulation, and confirmation simple.

5 Conclusion and perspectives

As we hope to have demonstrated in this paper, ETAP has now reached a state where it is suitable for both demonstration and experimentation. The project is available on GitHub (`github.com/didierverna/etap`), and as a matter of fact, we were quite happy that after the presentation at TUG 2023, half a dozen attendees immediately expressed some interest in using it. Consequently, we immediately updated the installation instructions so that everyone may now use it, without any prior knowledge of Lisp.

The existing list of planned improvements is already quite large. For example, more flexibility in font selection is a high priority. In general, our plans for the future will follow three complementary directions.

Bibliography We plan on studying more line-breaking literature and port existing ideas or algorithms we find to ETAP. By the way, here is a small plea for help: our Duncan [6] and Barnett [2] implementations are based only on the descriptions that are given in the Knuth-Plass paper. We couldn't recover the original publications, and hence would love it if anyone could contribute them.

Research One of our top priorities in the research area is working on river detection. We also plan on looking into microtype extensions, and perhaps a number of smaller or simpler issues.

One such issue is what we call "character ladders". In the same way TeX has this notion of "double hyphen demerits" for hyphenation ladders, it is usually undesirable that consecutive lines begin or end with the same characters or short words. Taking this into account is in fact very simple, and can even be implemented as an extension to the Knuth-Plass algorithm by adding a new kind of demerits.

Speaking of Knuth-Plass extensions, the ability to have a graph-based implementation as well as the original dynamic programming optimization opens the door to a number of interesting research questions. For example, the way TeX addresses adjacency problems is rather coarse. It only has four fitness categories because in order for its cost function to remain dynamically optimizable, the number of active nodes to keep around depends on the number of fitness classes (which needs to be discrete!). This is in fact sub-optimal because if two consecutive lines belong to different classes, the adjacency cost will be the same, whether or not those lines are close to each other in terms of scaling. On the other hand, if one maintains a full graph of possible breaking solutions, the adjacency demerits can be turned into a continuous, hence much more accurate function. It would be interesting to see how much of a difference this makes in practice.

Development Finally, there are also some more technical aspects that we want to address, one of them being a tighter integration between the GUI and the platform's core. The current design offers a number of features that are not yet accessible to the GUI. For example, every possible break point in the lineup has a local penalty value that can be changed programmatically. The GUI only allows global changes to the default initial value (TeX's hyphen penalty for example), but it would be nice if we could, say, right click on hyphenation points, and get a slider to change said penalty, having the paragraph reformatted immediately. Indeed, this would be an extremely convenient feature to have in a production system as well.

References

[1] ANSI. American National Standard: Programming Language — Common Lisp. ANSI X3.226:1994 (R1999), 1994.

[2] M.P. Barnett. *Computer Typesetting: Experiments and Prospects.* MIT Press, Jan. 2000.

[3] K. Bazargan. Batch commander: a graphical user interface for TeX. *TUGboat* 26(1):74–80, 2005. `tug.org/TUGboat/tb26-1/bazargan.pdf`

[4] R.S. Bird. Transformational programming and the paragraph problem. *Science of Computer Programming* 6(2):159–189, 1986. `doi.org/10.1016/0167-6423(86)90023-7`

[5] R. Burstall. Christopher Strachey — Understanding programming languages. *Higher Order Symbolic Computation* 13(1–2):51–55, 2000.

[6] C. Duncan, J. Eve, et al. Computer typesetting: an evaluation of the problems. *Printing Technology* 7:133–151, 1963.

[7] A.C. Kay. *The Reactive Engine.* Ph.D. thesis, University of Utah, 1969.

[8] D.E. Knuth. *The TeXbook.* Addison-Wesley, 1984.

[9] D.E. Knuth. *TeX: The Program*, vol. B of *Computers & Typesetting.* Addison-Wesley, Jan. 1986.

[10] D.E. Knuth, M.F. Plass. Breaking paragraphs into lines. *Software: Practice and Experience* 11(11):1119–1184, 1981. `doi.org/10.1002/spe.4380111102`

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

[12] M.D. McIlroy. Macro instruction extensions of compiler languages. *Communications of the ACM* 3:214–220, Apr. 1960. `doi.org/10.1145/367177.367223`

[13] O. de Moor, J. Gibbons. Bridging the algorithm gap: a linear-time functional program for paragraph formatting. *Science of Computer Programming* 35(1):3–27, 1999. `doi.org/10.1016/S0167-6423(99)00005-2`

[14] J. Stoy, C. Strachey. OS6 — An experimental operating system for a small computer. Part 2: Input/output and filing system. *The Computer Journal* 15(3):195–203, 1972.

[15] D. Verna. Star TeX: the next generation. *TUGboat* 33(2):199–208, 2012. `tug.org/TUGboat/tb33-2/tb104verna.pdf`

[16] D. Verna. ETAP: Experimental typesetting algorithms platform. In *15th European Lisp Symposium*, pp. 48–52, Porto, Portugal, Mar. 2022. `doi.org/10.5281/zenodo.6334248`

◇ Didier Verna
EPITA Research Lab
14–16, rue Voltaire
94270 Le Kremlin-Bicêtre
France
`didier (at) lrde.epita.fr`
`https://www.lrde.epita.fr/~didier/`
ORCID 0000-0002-6315-052X

Living in containers — on TEX Live (and ConTEXt) in a Docker setting

Island of TEX

Abstract

Over the course of the last year(s), the Island of TEX has received quite some interest in its Docker containers. This article gives a brief overview about our container infrastructure for TEX Live and ConTEXt, including some examples on using our containers in production environments. Last but not least, we will elaborate on some interesting (mostly still open) problems connected to containerizing TEX Live.

1 Overview of the Island's Docker images

Since 2019, the Island of TEX provides multiple Docker images for TEX Live and ConTEXt. Our first publication on this topic was in *TUGboat*, Volume 40 (2019), No. 3 and stays mostly relevant. Therefore, let us keep this introductory section short.

We still conceive Docker as an easy way to ship a portable setup of software to users by providing them with an operating system layer, operating system packages (like Python) and the software layer (in our case TEX Live or ConTEXt LMTX) bundled into one single compressed file (the Docker image; simplified but sufficient for visualization). By pulling a Docker image the user has a reproducible setup at hand without caring about his own host operating system or software dependencies.

To this end, we provide a fairly minimal ConTEXt LMTX image (`contextgarden/context:lmtx`) with the LMTX standalone distribution and the modules that ship with this ConTEXt distribution. The image is based on Debian (testing branch) and is about 250 MB in size.

On the other end of the spectrum we provide Docker images for TEX Live (`texlive/texlive`) in different flavors. For all releases from 2013 on up until the latest historic release (currently 2022), we provide the historic images following the naming scheme `TL{YEAR}-historic`. They always contain a `scheme-full` TEX Live installation without documentation and source tree unless you explicitly request one or the other of these removed trees by appending `-doc` or `-src` respectively (`doc` first if you want to have both, e.g. `TL2022-historic-doc-src`).

Like the ConTEXt images, the historic images are based on Debian's testing branch. Additionally, they ship with required software to run most of the tools included in TEX Live, e.g. Java for tools like `arara`, Python for `pygments` (needed for the popular `minted` package), and so on. This comes at a cost: the `TL2022-historic` image without documentation and source files tips the scales at 2.16 GB.

A note on the word `historic`: The images are structured into multiple layers, one of them being the historic TEX Live tree that does not change. However, the other layers which contain the operating system and OS software packages are updated monthly. This does not necessarily reflect the operating system software situation that has been present in the historic TEX Live release's year but is beneficial from multiple other points of view, e.g. when you want to run your own scripts for pre- or postprocessing.

Apart from the historic images, we provide images for the latest release of TEX Live (currently 2023). The basic setup concerning the Debian base and software packages is identical to the historic images. However, in addition to splitting off the documentation and source tree you may request any of the TEX Live schemes of the latest release by appending a hyphen and the scheme's name to the `latest` tag, e.g. `latest-small` or `latest-medium-doc`, to name just two variations. The default `latest` tag will pull a `scheme-full` installation: handle with care. All these various images are updated weekly, with both Debian and TEX Live package updates.

You can find our images on both Docker Hub:

`hub.docker.com/r/texlive/texlive`
`hub.docker.com/r/contextgarden/context`

and our GitLab registry; see the projects' code repositories at:

`gitlab.com/islandoftex/images/texlive`
`gitlab.com/islandoftex/images/context`

2 Using the images in a local setup

One of the two primary use cases we focus on when developing the images is use in a local environment, i.e. replacing your local TEX Live installation. To start off, let us emphasize that usually, especially when using the latest TEX Live release locally, you do not want to use our Docker images here for various reasons including imperfect updating strategies and space overhead.

However, in many settings there are a number of benefits using Docker images locally, especially when using the historic images. For the sake of this discussion, let's suppose you are coordinating a team who have used LATEX for their various documents on various operating systems for years.

. . . enter story telling mode . . .

Now let's imagine a new, not so tech-savvy, contributor joining; meet Bob. You have to explain to him how to install a TEX installation on his machine which runs an operating system you are not comfortable interacting with. You use last year's

doi.org/10.47397/tb/44-2/tb137island-docker

```
image: registry.gitlab.com/islandoftex/images/texlive:TL2022-historic

build:
  script:
    - find -name "*.tex" -exec arara -v "{}" \;
  artifacts:
    paths:
      - ./**/*.pdf
```

Figure 1: Preliminary `.gitlab-ci.yml` file to rebuild all `.tex` sources with `arara`.

TeX Live release so you have to refer Bob to one of the guides on how to install a historic TeX Live release there. And you use `arara` for build management and `minted` in your documents … the thought of guiding Bob through Java and Python installations and debugging a setup on another operating system is not that appealing. But wait a minute, at this point you could also refer him to one of the various setup guides for Docker and let him `docker pull texlive/texlive:TL2022-historic` and be done.

No sooner said than done. Bob now has his Docker-based TeX Live up and running, including all the dependencies needed for his daily typesetting. He creates his first document and runs it in the Docker container using:

```
docker run -i \
  -v "$PWD":/opt/doc:z -w /opt/doc \
  texlive/texlive:TL2022-historic \
  arara -v document.tex
```

To avoid typing all this every time, he configures his editor to run this command on compilation. Bob is happy, you are happy, onboarding done.

As a short interlude for the interested reader: the longish command above pulls and runs the historic image of TeX Live 2022 from Docker Hub (remember that it is the image without documentation or sources). When it starts the image, it will mount the current working directory in the Docker image, ensuring that all documents are accessible and the build results will appear there. It then starts the `arara` call at the end of the command, in interactive mode so you can interact with the output of the command transparently through the Docker boundary.

A week later, Bob is tasked to typeset an updated version of one of the older documents which does not compile on TeX Live 2022 any more. To see how it looked back then, an older TeX Live release is needed. Luckily, you have avoided needing to remind him of the installation instructions for a historic TeX Live release just to find yourself in the situation to explain to him how to set the `PATH` variable of his operating system to the older release and back to the newer one. You just let him duplicate his editor configuration for TeX Live 2022 and use the older release.

```
docker run -i \
  -v "$PWD":/opt/doc:z -w /opt/doc \
  texlive/texlive:TL2018-historic \
  arara -v document.tex
```

and everything works. Docker even pulled the image on first use without needing a separate pull command.

Now that you just happened to have finished onboarding a new contributor you decide to write a short setup guide for future new contributors in your team. Interestingly enough, the whole setup guide fits onto one A4 page. Happy that you have a concise guide covering everything from installing to running multiple TeX Live releases with a reproducible environment and dependencies, you close that onboarding chapter.

3 Using the images in a CI setting

Bob got hooked, all this Docker business was easy enough to be well hidden behind his editor for now. All this technical stuff being a bit magical to him he would still like to verify that documents he makes available to you will always compile. As you are using GitLab anyway, you introduce him to `git` (a lot more work than the one A4 page for the local setup) and set up a continuous integration pipeline on your GitLab instance that compiles all the documents when a new commit is pushed.

The setup is simple: you add a `.gitlab-ci.yml` file to your repository which has the content shown in figure 1.

Done. The pipeline runs and finishes … after quite some time. Waiting 10 minutes for the feedback that the documents Bob just touched compile seems a bit subpar to you. Your inner Don Knuth starts yelling at you about something with premature optimization but you are convinced: the repository grows, it cannot be a good idea to always run all documents when we are only interested in potential compilation errors of a few of them.

So your preferred setup would instead look something like this:

```
...
  script:
    - bash compile-only-needed.sh
...
```

with some bash magic taking care of compiling only what is needed. A deep dive into GitLab's documentation later you see this is not as hard as you had imagined. So your bash script is surprisingly short:

```
#!/usr/bin/env bash

gitsha="$(curl \
  --header "PRIVATE-TOKEN: \
           $GL_API_ACCESS_TOKEN" \
  "https://gitlab.com/api/v4/projects/\
$CI_PROJECT_ID/pipelines?ref=$CI_DEFAULT_BRANCH\
&sort=desc&status=success" \
  | grep -o -E -m1 '"sha":"([^"]*)"' \
  | head -1 | cut -c 8-47)"

changed_files=$(git diff-tree \
  --no-commit-id --name-only -r \
  "$gitsha".."$CI_COMMIT_SHA")
compile_all=false
for file in $changed_files; do
  if [[ $file == texmf/* ]]; then
    compile_all=true
    break
  fi
done

if [ "$compile_all" = true ]; then
  latex_files=$(find . -name "*.tex")
  for file in $latex_files; do
    if grep -Fq "arara" "$file"; then
      base_dir="$(dirname "$file")"
      base_name="$(basename "$file")"
      cd "$base_dir"
      arara -v "$base_name"
      cd "$(git rev-parse --show-toplevel)"
    fi
  done
else
  for file in $changed_files; do
    base_dir="$(dirname "$file")"
    base_name="$(basename "$file")"
    if [[ ! -f "$file" ]] \
    || [[ "$file" != *.tex ]] \
    || ! grep -Fq "arara" "$file"; then
      continue
    fi
    cd "$base_dir"
    arara -v "$base_name"
    cd "$(git rev-parse --show-toplevel)"
  done
fi
```

You know that you ignored most of the sanity checking you should have done. But as the old engineering adage says: "it works". It even takes into account that it needs to recompile everything if one of your "global" files changes — the ones you have in your `texmf` folder in your repository like your logo, custom packages, etc.

The basic structure is even easy to explain: first, the GitLab API is queried for the last commit on your default branch a pipeline has successfully run. Then, `git` is run to determine all files that have changed since that commit. If one of the changed files affects all documents, the CI runs basically the `find` call of your first CI example[1] with some directory changes. If no such file has been changed, `arara` is run on all relevant changed LaTeX sources.

With this simple bash script, you successfully turned your 10 minute pipeline into a 3 minute on average pipeline leaving you quite satisfied but wondering why it takes so long to typeset one or two documents.

A short investigation of the CI log later you have identified the culprit: you use the full TeX Live image, i.e. `scheme-full` which pulls more than 2 GB each time your pipeline runs, making up more than 2 minutes of that 3 minutes. Unfortunately, you are using a historic image which does not provide a split by schemes (and you use too many packages anyway) so you cannot slim down on that one. But you notice that your team has spare server capacity and set up a GitLab runner for your project that caches the historic TeX Live images.

Now that you have successfully reduced your average pipeline to less than one minute running time you are confident that it is future-proof enough. And after a few more minutes than originally intended you have successfully implemented Bob's request.

. . . story telling mode off . . .

4 Maintenance challenges of the TeX Live images

The Island of TeX manages and builds all its Docker images using GitLab and the GitLab CI. Unfortunately, we reached some limits on the main instance at `gitlab.com` quite early in our image build processes. Thanks to Marei Peischl and Vít Starý Novotný we have access to custom CI runners (read: servers that build our images) which have massively improved the stability of our build process.

[1] At this point we should add that that `find` call in the first example would not work properly due to executing `arara` in the wrong base directory, which is a bad idea in general. But as it was a motivating example and is fixed in the bash script, we consider this error a case of "no harm done".

However, there is still something calling for manual intervention every other week. So we are interested in improving our build setups to avoid all this intervention. Part of that will include switching to new infrastructure, and part of that in turn will include optimizing the build process and caching.

To further build optimizations, we would like to provide the historic images split by scheme as we do for the latest images. This will require more substantial changes than we would like to admit but also bring the benefits of smaller images to many, especially as we acknowledge the importance of the historic images.

Revisiting the topic of automation, there is a minor annoyance also caused by requiring manual intervention: each year, when a new TEX Live is released we have to add the now-historic TEX Live to our build matrix for historic images. We would like to fix this but have not found the way to go yet. Ideas are welcome.

Another topic that looks for helping hands is the layering of the TEX Live images. This is especially important with the split by schemes which could potentially be layered on top of each other but also to improve the update situation for local uses of the latest images. Experimentation, ideas, and fruitful discussions on our issue tracker at `gitlab.com/islandoftex/images/texlive/-/issues` would be greatly appreciated.

Last but not least, there is one topic that has been a challenge so far but is on the short-term roadmap of actually being resolved: providing multi-architecture images. Currently, our images only provide binaries for the `x86_64` architecture but a few platforms, most notably smartphones and Raspberry Pis, run on a different architecture, namely ARM. TEX Live ships with binaries for these architectures and by the next time you hear from us we hope our images do too.

◇ Island of TEX
https://gitlab.com/islandoftex
https://islandoftex.gitlab.io

News from the HINT project: 2023

Martin Ruckert

Abstract

The HINT file format [5] was presented at TUG 2019 [4], and at TUG 2020 [6], the first usable viewer for HINT files was presented. The HiTEX engine became part of TEX Live in 2022. This presentation will explore the changes that have taken place since then and what to expect in the future. This article will

- explain the improvements in glyph rendering in more recent versions of the HINT file viewer;
- describe the use of links, labels, and outlines;
- present hints on how to design TEX macros for variable page sizes;
- and discuss the capabilities of the HINT file format to convert pages to plain text for searching or text-to-speech processing.

1 Displaying glyphs

Initially, the HINT viewer supported only `.pk` fonts. These font files contain METAFONT fonts at a fixed resolution, usually 600 dpi. Rendering such a font on a computer screen with a typically much lower resolution, was done in three steps:

1. Decoding the font file header and caching it for later use.
2. Decoding a glyph into a black and white bitmap and caching it for later use.
3. For each pixel on screen intersecting the glyph's bounding box
   - map the pixel center to a source point in the glyph's bitmap, and
   - compute the pixel's gray value by linearly interpolating the black and white values of the four pixels surrounding the source point in the bitmap.

Since high resolutions, even above 300 dpi, are common on small mobile devices, the results were more than acceptable on these devices. On ordinary computer screens, typically with resolutions less than 100 dpi, the results were insufficient. In particular, the rendering of thin lines would distribute the available amount of black ink over a two-pixel-wide area and the line would fade away into a blurry light-gray.

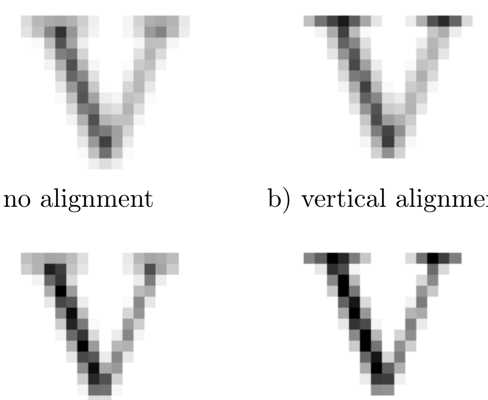
Things changed with the use of the FreeType font rendering library [7]. This library can render PostScript Type 1 outline fonts at any resolution desired. After replacing the `.pk` fonts by `.pfb` fonts, the viewer could render the glyphs as gray-value bitmaps for the actual screen resolution [3]. To produce good looking glyphs from an outline font, first the

Martin Ruckert
doi.org/10.47397/tb/44-2/tb137ruckert-hint23

positions of key points of the outline, for example the points where the outline has a horizontal or vertical tangent, will be rounded to the pixel grid. After that, pixels that are only partly covered by the outline will be assigned gray values, depending on the amount of coverage. This results in less blur and consistent stroke widths, improving readability especially for small font sizes.

The quality of the font rendering in the HINT viewer was, however, still inferior to a rendering of the same font by other programs. The reason was that the viewer would not map the glyph bitmap one to one to the screen but instead would map the bitmap to TeX's exact glyph position — usually not aligned to the pixel grid — using step 3 as given above.

To improve readability at small font sizes, the current viewer rounds the glyph position to the pixel grid before rendering the glyph. It also replaces the linear interpolation of pixel values by using the gray value of the nearest source pixel. The rounding will occur only if the font size is below a given threshold. In principle the rounding can be split into rounding horizontal and rounding vertical position. While the first affects character distances, the latter moves entire lines and is less distracting. For a demonstration, see [3].



a) no alignment b) vertical alignment

c) horizontal alignment d) full alignment

Figure 1: A cmr 10pt V with different alignment to the pixel grid.

Further improvements are possible, but not yet implemented. One potential method is oversampling, where a glyph is rendered at, for example, four different horizontal positions on the pixel grid. Choosing one of these four renderings, the horizontal glyph position must be rounded to 1/4 of the pixel size which is far less distracting. Another method is sub-pixel rendering. This method uses the fact that one white pixel on screen actually consists of three colored dots: red, green, and blue. So by considering them as independent light sources, the horizontal resolution can be tripled. This improves the positioning but leads to colored borders which some people find distracting.

2 Links, labels, and outlines

People my age learned how to navigate through thick books already in primary school, if not in kindergarten. These skills are more or less obsolete when it comes to navigating through "thick" electronic documents. So good replacements are necessary. The most obvious point to start exploring a book is its table of contents where for each section the corresponding page number is listed. The HINT file format supports the concept of a home page: a position in the document identified by the author that can be reached in the viewer with a single key stroke, touch, or click. The HINT document, however, has no fixed page numbers. The pages grow and shrink with the window size (and with the magnification factor). So instead, a table of contents must use a clickable link that brings you immediately to the section in question. Similar links are used for the table of figures, index, and all kinds of cross-references, be it to individual parts of the text, a figure, a table, a citation, or a displayed formula.

As an alternative to the table of contents, the HINT file format also supports "outlines": A clickable table of contents, hierarchically organized and displayed in a separate window. To allow optimal use of the available space, sub-levels of the hierarchy can be hidden or expanded as needed [3].

At present, a driver [2] for the LaTeX `hyperref` package offers support for most of the above features.

In one respect HINT files are radically different from books and PDF files: There are no predefined pages. So following a link is not as simple as displaying a page with a given page number, but requires finding two good page breaks so that the target is on the page between them. The algorithm used in the current HINT viewer is still under development and there are cases where the choice of page breaks could be better.

3 Designing macros for variable pages

The traditional implementation of centering text is the `\centerline` macro. It expands to `\hbox to \hsize{\hfill` *text* `\hfill}` which will look nice as long as the *text* is shorter than `\hsize`. If the text is longer, it will produce an overfull box, stick out into the margin, and even go past the edge of the window. A better solution uses TEX's line breaking procedure which requires a vertical box.

```
\vbox{\rightskip 0pt plus2em
      \leftskip=\rightskip
      \parfillskip=0pt\parindent 0pt
      \spaceskip.3333em
  \xspaceskip.5em\relax
This is Text Centered on the Page
}
```

Letting `\rightskip` and `\leftskip` stretch enough, but not too much, so that the line breaking routine will try to keep the lines filled but still has enough room to produce decent lines (see [3]). The interword glue, on the other hand, is prevented from stretching. It could be made to allow some shrinking to gain additional flexibility.

The only new feature introduced in HiTEX since 2019 is support for `\vtop`. This is important because writing for variable page sizes often requires replacing a horizontal box by a vertical box to enable the breaking of paragraphs into lines. `\vtop` is required if multiple vertical boxes need to be aligned on the top baseline (see [3]).

4 Searching

The user input in a search field is just a plain sequence of characters coded in UTF-8 or perhaps another encoding such as ISO 8859-1. The text as represented in a TEX document is far more complex and searching requires finding a match between both representations. Even if the input consists only of ASCII characters the HINT viewer must handle some special cases.

If the word the user wants to find uses a ligature, the match is made using the replacement characters, which are retained in TEX's ligature node. If the word on the page is hyphenated and split across lines, the match must ignore extra characters inserted by the pre- and post-hyphenation lists, as well as the space that is usually separating the word at the end of one line from the word that starts the new line.

Thus, the HINT backend provides a function that converts entire pages into sequences of characters moving from top left to bottom right, eliminating the effects of ligatures and hyphenations and condensing various combinations of glue — indentations, spaces, baseline skips, left skips, and right skips, to name just a few — to a single space. Kerns, meanwhile, are completely ignored. An infelicity here is the definition of the LATEX macro, which uses a glue instead of a kern between 'A' and 'T'. So you have to search for "`LA TEX`".

It is planned to use the page-to-string function also to feed a text-to-speech converter.

Currently searching does not work well with non-ASCII characters, but it is planned to implement UTF-8 as the default encoding used for HiTEX and HINT files.

5 New viewers for Linux, macOS, and iOS

Together with the viewers for Windows and Android, the applications for Linux, macOS, and iOS complete the set of viewers. The Windows application, being the oldest and my workhorse for conducting experiments, is the most complex. The application for macOS is the most recent and was presented on Jonathan Fine's TEX Hour [1, 3]. The application for Linux is the simplest: it consists beside the backend and the OpenGL renderer (shared between all applications) of only a 600-line main program [2]. This is a good starting point for writing your own viewer.

References

[1] J. Fine, M. Ruckert, et al. Rethinking TEX in STEM. `texhour.github.io/2022/09/29/rethink-tex-in-stem/`, Sept. 2022.

[2] M. Ruckert. HINT source repository. `github.com/ruckertm/HINT`.

[3] M. Ruckert. The HINT video collection. `hint.userweb.mwn.de/hint/video/`.

[4] M. Ruckert. The design of the HINT file format. *TUGboat* 40(2):143–146, 2019. `tug.org/TUGboat/tb40-2/tb125ruckert-hint.pdf`

[5] M. Ruckert. *HINT: The File Format*. 2019. ISBN 1-079-48159-1. `amazon.com/dp/1079481591`

[6] M. Ruckert, G. Socher. The HINT project: Status and open questions. *TUGboat* 41(2):208–211, 2020. `tug.org/TUGboat/tb41-2/tb128ruckert-hint.pdf`

[7] D. Turner, W. Lemberg, et al. FreeType. `www.freetype.org/`.

◇ Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München
Germany
`martin.ruckert (at) hm dot edu`

Bumpy road towards a good LaTeX visual editor at Overleaf

Ben Davies

Abstract

Overleaf has both a Code editor and a Visual ("Rich Text") editor. We recently redesigned the Visual editor. Benefits, drawbacks, and specific issues this editor duality poses will be presented, together with some takeaways we have learned on the way.

1 Introduction

Overleaf is a collaborative tool designed to help people work together on a document. We do this by placing one 'true' version in the cloud which everyone works from. However, each collaborator will have different needs, so will need tools to suit them.

For instance, someone working on a document might not know LaTeX or is providing proofreading services and thus can do without seeing the raw form. It is important that we can enable these people to contribute content and make changes to the document without feeling excluded. Further, we want to reduce the burden that can sometimes be placed on those who do know LaTeX in such circumstances.

Another place we hope to make a difference is in learning LaTeX and for those intimidated by code. We want to help lower the barrier to learning LaTeX by making it more familiar and easier to interact with. In these cases, a mode that focuses just on the content but still allows edits to be seen without recompiling would seem to be a useful tool.

2 Road to the Visual Editor

A first attempt at such an editor was our Rich Text mode. This used CodeMirror 5 to decorate parts of the document such as maths (using MathJax) and figures. There were also attempts at 'hiding' code that didn't need to be seen all the time, such as the preamble and common styling commands. Regardless of the decorations, a core principle ensuring that code was always accessible was, and is, maintained.

However, at this time, the Source mode used the Ace editor; as a result, certain features were not available across both modes. Collaborative features such as track changes and commenting were not transferable to the Rich Text mode, limiting collaborators' ability to contribute. This meant people needed to switch back and forth between the modes costing them time and causing distraction.

Given these issues, we decided to migrate both editors to CodeMirror 6. There are several advantages to this outside of the Visual editor, such as better support for accessibility features, mobile devices and non-Latin languages. A key feature is the ability to support both versions of the editor: Code and Visual.

After the migration, we were able to share features across both editors more easily, bringing the collaborative features to the Visual Editor. Not only that, there is now parity between the editors when it comes to, for instance, themes, keybindings and auto-completion. This is because the Visual Editor is now effectively the Code Editor with decorations.

Having completed the migration we are now using an element-by-element design approach to improve the experience. Focusing on specific environments or commands allows us to make more continuous improvements and helps us consider the best behaviours more fully.

3 Features

The best way of experiencing the Visual Editor is to go to Overleaf, start a project and toggle the Visual Editor on. You will find a toolbar containing common actions that also provides familiarity for those new to LaTeX. There's a figure modal so you can insert figures without writing any LaTeX at all.

Given the documents that are usually written in LaTeX it is perhaps unsurprising that the maths environments are also decorated (now including custom maths commands) as well as decorated headings, lists, theorems, text styling and much more.

The Visual Editor was demonstrated at the conference; the video of the talk is available at `youtube.com/c/texusersgroup`.

4 Conclusions

We still have questions about the content we can handle efficiently and how certain things should behave, especially across the two editors. There are also considerations about how much can be done whilst ensuring we retain the full power of LaTeX for all users.

We did learn that providing parity to different code editors (CM6, CM5, Ace) is difficult; we are happier with one. We also learned that just because we know how we would write the LaTeX doesn't mean we know what the 'button' should do and *vice versa*. And finally that providing different interfaces enhances user experience and has been met very positively by the community. So long as we are achieving that, we will keep at it!

◇ Ben Davies
Overleaf
`ben dot davies (at) overleaf dot com`

doi.org/10.47397/tb/44-2/tb137davies-visual

Overleaf and TeX Live

Tom Hejda

Abstract

Overleaf makes an annual deployment of TeX Live, which we sum up here, including the testing we perform as part of the deployment process. The talk at the TeX Users Group conference was followed by a discussion about Overleaf's process with regards to LaTeX development and TeX Live testing; however, this discussion is not captured in this article.

1 Introduction

Overleaf is an online LaTeX collaborative platform that is available at `overleaf.com`. For more background on TeX Live, see `tug.org/texlive`.

The LaTeX compiler is run as a Docker image that contains a modified version of the `texlive-full` scheme. In this short article, the actions needed to successfully deploy new TeX Live images will be presented.

2 TeX Live deployment procedure

We deploy TeX Live usually in the third quarter each year. The procedure can be summed up in the following steps:

1. Prepare the initial Docker image that contains a full Linux installation and has `texlive-full`.
2. Use `tlmgr` to update the packages to the latest versions.
3. Make sure that helper tools such as ImageMagick, Inkscape, and requested R packages are properly installed.
4. Optimize fonts available in the Docker image — remove duplicates of fonts coming from multiple sources, and precompile fonts.
5. Remove the documentation that was installed together with `texlive-full` to decrease the image size.
6. Perform testing (see next section for details) and write documentation.
7. Go live and monitor for further issues reported by users.

3 Testing of the TeX Live image

Overleaf is running the Overleaf Gallery, which currently contains about 10 thousand LaTeX templates and example documents. With each new TeX Live version, we check whether the templates compile under the new version; the goal is to make each template use by default the most recent version possible, and maximize the number of templates that can run on the new version.

To this end, we manually check the templates that fail with the latest TeX Live version; sometimes it is possible that a simple patch to a package would solve the issue, in which case we try to coordinate with the package maintainers and see whether a fix is feasible. If that is not possible, we keep the template at an older version to ensure it uses a version where it runs without errors.

4 Conclusion

We are always looking for improvements to the process; currently we are aware of the issue of bad alignment in timing between our process and the TeX Live annual build procedure, and we are looking into ways of improving this while still giving our users good and stable experiences with the compiler.

The video of the talk is available at `youtube.com/c/texusersgroup`.

◇ Tom Hejda
Overleaf
`tom dot hejda (at) overleaf dot com`

doi.org/10.47397/tb/44-2/tb137hejda-overleaf-tl

Primo — A new sustainable solution for publishing

Rishikesan Nair T, Apu V, Hàn Thế Thành, Jan Vaněk

Abstract

Primo is a cutting-edge, cloud-based authoring, submission, and proofing framework that provides a sustainable solution for academic publishing. It combines the advantages of XML-based workflows that facilitate controlled authoring and/or editing in accordance with specific DTDs and house styles, with the visually appealing and mathematically precise typesetting language of TEX, enabling the creation of high-quality PDFs and mathematical images (offering an alternative to MathML coding).

By speaking the widely accepted communicating lingua franca of mathematics and science (i.e., TEX), and utilizing the XML/MathML format for archiving, Primo has the potential to revolutionize the publishing industry. This tool caters to both the author and the publisher, bringing their needs together with enhanced participation of authors in the publishing process. The three main modules of Primo include Authoring, Submission/Reviewing, and Proofing, all of which are equipped with usability checks during submission, a collaborative editing feature, a WYSIWYG math editing tool, and publisher/journal-based PDF manuscript rendering. With Primo, authors can be assured that their work will be published with the highest level of precision and quality.

1 Introduction

Primo, the latest addition to the lineup of TEX-based tools, is developed by STM Software Engineering Pvt Ltd., a specialized TEX typesetting house renowned for its top-notch typesetting and prepress services, catering to the needs of STM publishing giants specialized in the typesetting of complex articles. With its state-of-the-art technologies, STM Software Engineering Pvt Ltd. developed a range of cloud-based typesetting frameworks, including TEXFolio [2] and Ithal [1], primarily designed for in-house typesetting and format conversion purposes within publishing houses. On the other hand, Neptune [3] and Primo target authors directly, providing them with efficient and user-friendly TEX-based tools.

2 Primo

Primo's modular structure and well-designed tools enable authors to navigate the entire publication journey with ease, from initial authoring to final proofing. By integrating these three modules, Primo optimizes the authoring, submission, and proofing processes, making it a comprehensive and efficient platform for scholarly publishing.

3 Authoring tool

An offshoot of the aforementioned processes is a standalone authoring tool, currently named Primo Editor (figs. 1–4). This tool encompasses all the necessary elements to compose an article meeting all technical requirements for seamless uploading to a publisher.

3.1 Salient features

Please note that while the list below covers the main features, undoubtedly additional features will be added.

1. **Collaborative editing:** Multiple authors can contribute simultaneously.
2. **Operating system independence:** The tool is operating system independent and has a WYSIWYG interface.
3. **Cloud-based with TEX installation:** A completely cloud-based version with a TEX installation comprising essential packages, fonts, compilers and utilities. The user need not worry about installation or configuration of a TEX distribution.
4. **Proper template:** With the Primo authoring tool, authors no longer need to spend time searching for the appropriate template for their submissions. The tool offers a comprehensive collection of templates, and given the name of the journal and publisher, it automatically selects the most suitable template. This eliminates the hassle of manually locating the correct template and ensures that authors can focus on their content without the added burden of formatting.
5. **Math input:** Users can enter mathematical equations using LATEX syntax or utilize built-in math tools from the menu.
6. **Form-like interface:** A user-friendly form-like interface is available to capture front matter data such as author information, affiliations, abstract, keywords, graphical or stereo-chemical abstracts, and more.
7. **WYSIWYG interfaces for tables and figures:** WYSIWYG interfaces for entry of table and figure data.
8. **BIBTEX support:** BIBTEX data is always welcome if the user prefers to use the same.
9. **Bibliography data checking:** Checking bibliography data with Crossref is an added benefit.

doi.org/10.47397/tb/44-2/tb137rishi-primo

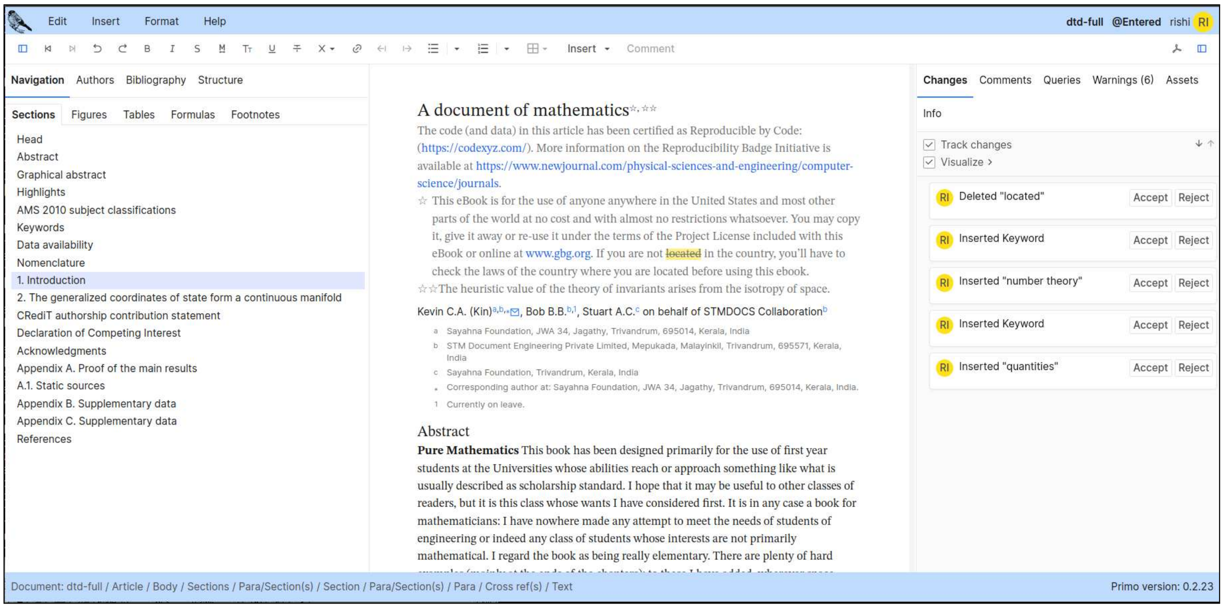


Figure 1: Primo: The main page.

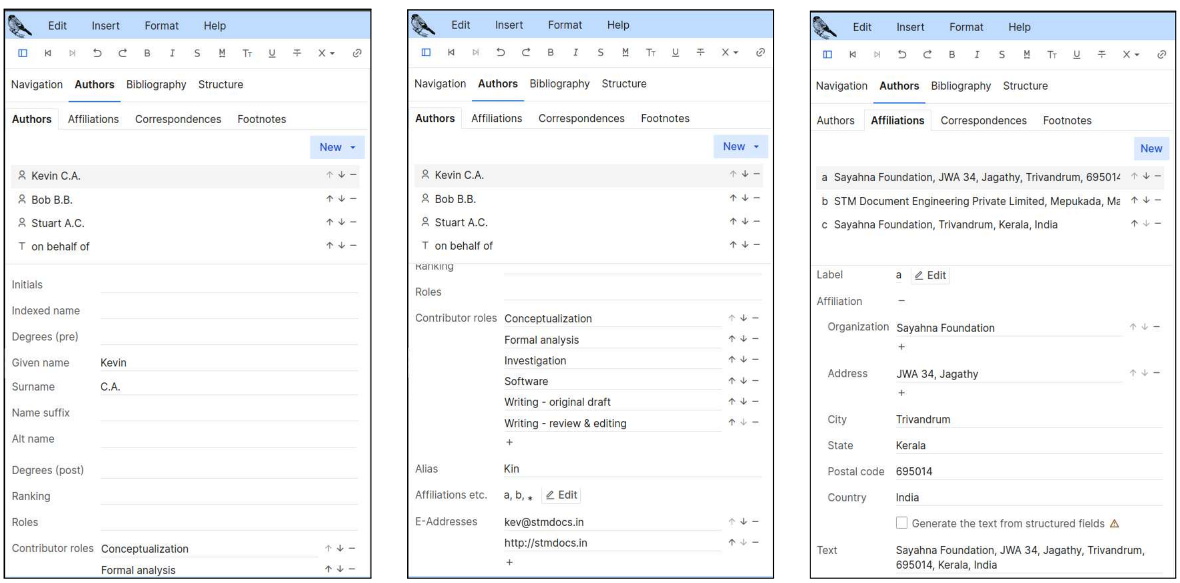


Figure 2: The author and affiliation screens.

10. **Bibliography import:** With the help of Primo, users can easily import bibliography data using identifiers such as DOIs (Digital Object Identifiers) or PMID (PubMed IDs), streamlining the referencing process.

11. **Enhanced author participation:** The tool promotes active author involvement in the publishing process, minimizing errors and semantic inconsistencies introduced by typesetters. This enhances the overall quality and reduces the time gap between submission and publication.

12. **Compliance checking:** The tool automatically checks the manuscript's adherence to the style guidelines of the publisher or journal, ensuring compliance with formatting requirements.

These features collectively provide a comprehensive and user-friendly platform for collaborative manuscript preparation, improving the efficiency and quality of the publishing process.

4 Submission tool

The submission process for authors can often be arduous and time-consuming. With strict timelines

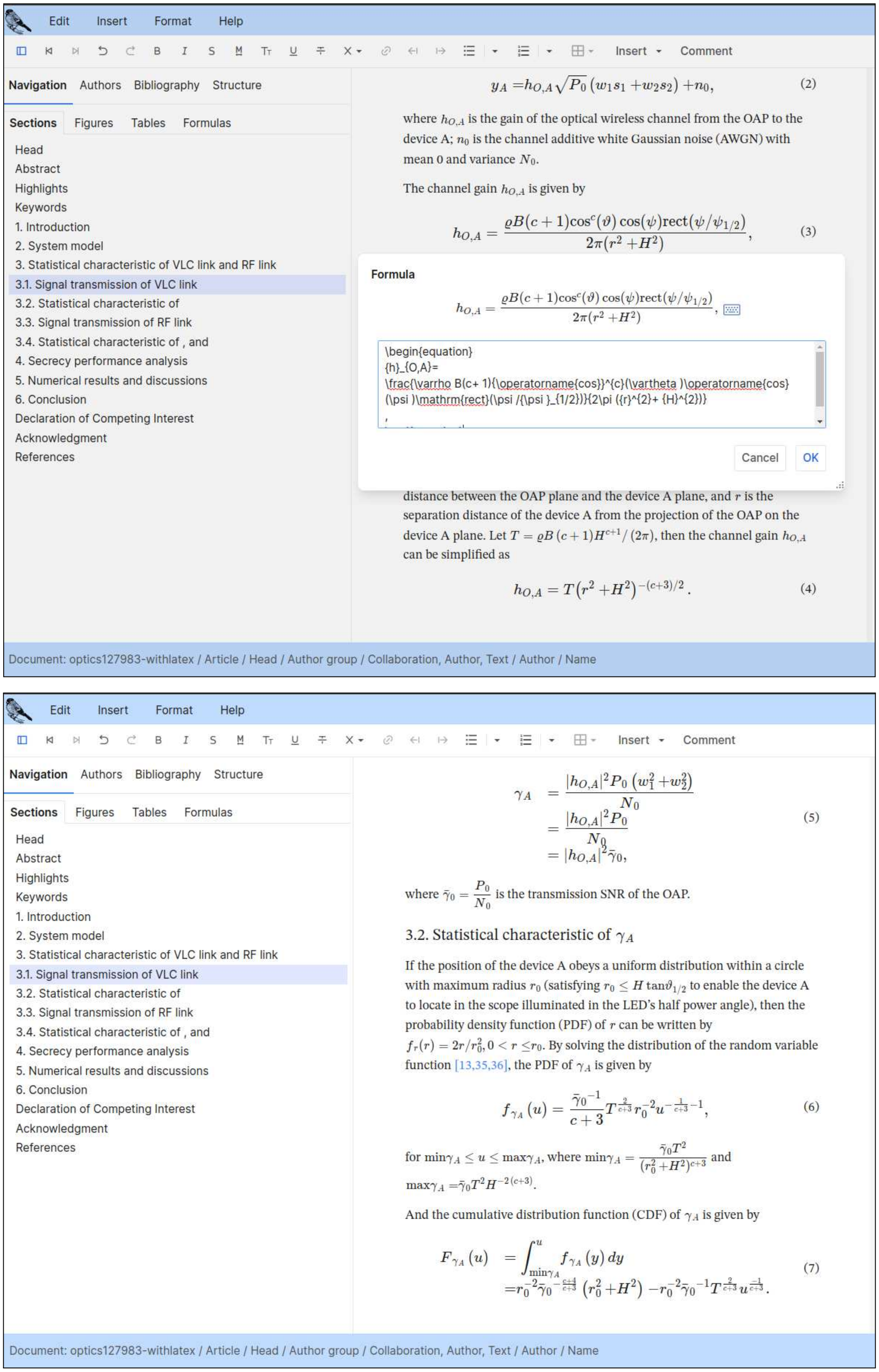


Figure 3: Math content rendering.

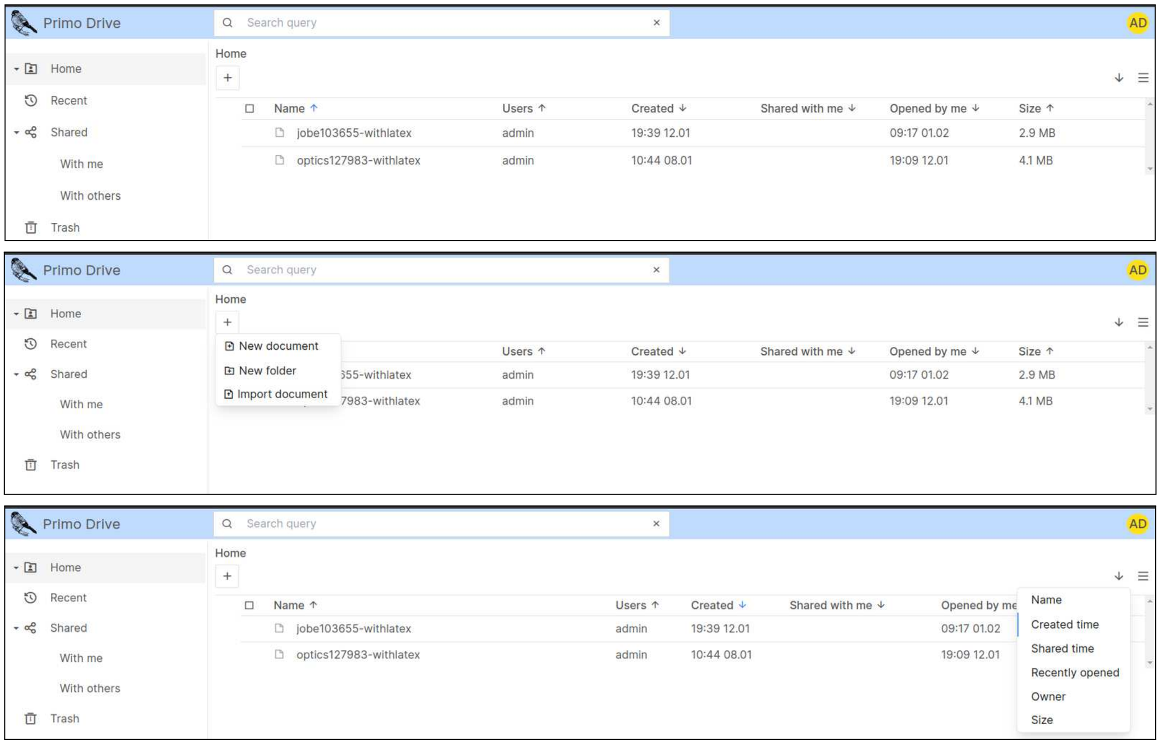


Figure 4: The Primo drive.

and numerous procedures to navigate, authors often find it challenging and frustrating. Primo seeks to alleviate these difficulties. In its initial stage, this authoring tool assists authors in crafting their manuscripts in compliance with the specific style requirements of publishers and journals, as described above. The subsequent step involves a seamless transfer of the source files to the submission system employed by publishers.

4.1 Salient features

1. **Source files:** Since the source files were already prepared as per the specification using the authoring tool, there should be no surprises in the submission system.
2. **Proper submission:** Transferring files directly from author tool to submission tool and finally to the publishers' submission system helps eliminate chances of missing files or materials.
3. **File category:** In the commonly-used submission systems, the file type of each source file uploaded has to be specified. For example, "Manuscript", "Revised Manuscript", "Figure", etc. Primo helps to sort this out easily and helps authors to identify which is which.
4. **Usability check:** Depending on the compliance of the submission system to which finally the source files are uploaded for the publisher, the Primo submission tool runs a custom usability check on the source files and reports problems any problems with the source files. This will help those authors who prepare manuscripts in their system or any other interface and directly upload source files to the Primo submission tool.

5 Proofing tool

The proofing tool plays a crucial role in Primo. Once the typesetting process is complete, the typesetter uploads or imports a dataset that includes the article's XML/MathML, figures, supplementary materials such as audio, video, program codes, and alternative images for MathML, among others.

The format of the dataset is simple and looks like this:

```
Archive:  ENDEND_99996.zip
  Name
  ----
  ENDEND_99996.pdf
  main.assets/
  main.assets/fx1006.jpg
  main.assets/gr1.jpg
  main.assets/fx999.jpg
  main.assets/gr2b.jpg
  main.assets/mmc1.pdf
  main.assets/fx1001.jpg
  main.assets/fx1004.jpg
  main.assets/fx1.jpg
```

```
main.assets/fx1005.jpg
main.assets/fx1002.jpg
main.assets/gr2a.jpg
main.assets/fx1003.jpg
main.assets/fx1007.jpg
main.pdf
main.stripin/
main.stripin/si33.svg
main.stripin/si121.svg
main.stripin/si165.svg
...
```

5.1 Salient features

1. **Track changes:** This facility helps the authors to understand the changes made by the typesetters in their document. The accept and reject buttons can be used to accept or reject the changes made by the typesetters. The visualize mode in the track changes has two further features: to visualize the changes made by the authors and to visualize the changes made by the copy-editors or co-authors.
2. **Comment:** If authors are unsure about how a change is to be done, then the comment facility can be used.
3. **Queries:** Queries raised by the typesetters or copy-editors are available in the query panel. Authors can provide replies below the queries.
4. **DTD compliant:** All and only the features allowed by the DTD can be used. For example, a figure cannot be inserted in the author field since the DTD does not support that.
5. **Changing the order:** Order of author names or position of given and surname or just the content can be swapped.
6. **Lists:** Changing the format of the list counters, or defining the format for new lists, is easy via a drop-down menu facility.
7. **Intelligent insertions:** Since the tool faithfully follows the DTD, the insert menu will show the items according to the content model only. So if you are in a paragraph, the drop-down list will show sections, math (both inline and display), list, display quote, etc. However, if you are in the reference section, the insert menu will show bib-entry, bibliography section, etc.

6 Technologies behind Primo

Primo is written mostly in Scala, both server-side and client-side. The client-side is compiled using Scala-JS to JavaScript. On the server-side, Scala is compiled to Java byte-code and runs in the JVM. It can seamlessly interoperate with existing Java libraries. The development environment is Intellij IDEA, the build-tool is SBT. Primo uses its own widget library called VDL, part of the Primo code base.

Primo doesn't have many external dependencies. We use following major libraries:

- the JDK, of course
- `undertow` — the web server, like Tomcat, but smaller
- `sqlite` — for the database
- `lucene` — full-text index of the documents

And some relatively minor libraries:

- `xpp3` — XML parser
- `scala-js` — DOM, Java time support, logging support, etc.
- `boopickle`, `scala-css`, and others

References

[1] Ithal. `ithal.io/main.html`

[2] Rishikesan Nair T., Rajagopal C.V., and Radhakrishnan C.V. TEXFolio — a framework to typeset XML documents using TEX. *TUGboat* 40(2), 2019, 147–149. `tug.org/TUGboat/tb40-2/tb125rishi-texfolio.pdf`

[3] Aravind Rajendran, Rishikesan Nair T., and Rajagopal C.V. NEPTUNE — a proofing framework for LATEX authors. *TUGboat* 40(2), 2019, 150–152. `tug.org/TUGboat/tb40-2/tb125rajendran-neptune.pdf`

◇ Rishikesan Nair T, Apu V
STM Document Engineering Pvt Ltd.
River Valley Campus,
Mepukada, Malayinkizh,
Trivandrum 695571
India
`rishi (at) stmdocs dot in`
`apu.v (at) stmdocs dot in`
`https://stmdocs.org`

◇ Hàn Thế Thành, Jan Vaněk
Trivic s.ro.
Družstevni 161, 763 15 Slušovice
Czech Republic
`thanh (at) trivic dot io`
`jan (at) trivic dot io`
`https://trivic.io`

Automated tagging of LaTeX documents — what is possible today, in 2023?

Ulrike Fischer, Frank Mittelbach

Abstract

The LaTeX *Tagged PDF* project was started in spring 2020 and announced to the TeX community by the LaTeX Team at the *Online TUG Conference 2020* [9]. This short report describes the progress and status of this multi-year project achieved with the LaTeX summer release 2023.

Contents

1 Introduction

A tagged PDF is a PDF with an additional semantic structure. The purpose of tagging and the technical background have been described in earlier articles [3, 4] and in the documentation of the `tagpdf` package [2] and won't be repeated here.

As a short motivation for how tagging improves the accessibility and the reuse of data, let's look at bank statements as an example. For a few months Ulrike gets them as tagged PDF[1] and can compare them with earlier untagged versions. The tagging of the PDF is quite basic: apart from a few tags around paragraphs it contains only a large table with three

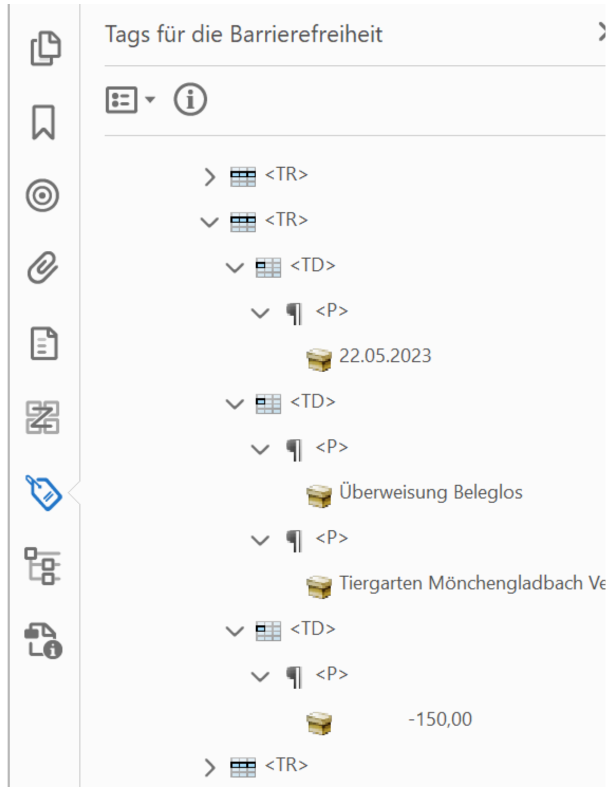


Figure 1: A tagged bank statement

Figure 2: Copy & paste of a tagged table into a spreadsheet

columns (the date, a multiline description and the amount). As shown in figure 1, the tagging adds `TR` (**T**able **R**ow) tags around rows and `TD` (**T**able **D**ata) around the cells.

If such a bank statement is read with a screen reader, then with an *untagged* PDF the reading order is messy: the amount is read somewhere in the middle of the multiline description, text surrounding the table can leak into the reading, and there are no options to navigate in the table. With a *tagged* PDF the experience is quite different: the reading order is correct, the screen reader announces row and column numbers and it is possible to navigate by rows and cells. Copy & paste from a tagging aware PDF reader improves too: while with an *untagged* PDF the table content is dumped into one column of a spreadsheet, the content of a *tagged* PDF is correctly split into single cells, as can be seen in figure 2.

[1] Their tagging is not done with LaTeX.

doi.org/10.47397/tb/44-2/tb137fischer-tagging23

2 Goals of the Tagged PDF project

The example hopefully shows why tagging is generally important and the LaTeX team embarked on the task to support tagging. To address a common misunderstanding: The main goal of the Tagged PDF project is not to make tagging *possible* — this is already the case today: if you want to produce a tagged PDF with LaTeX you can do it. The main goal is to make tagging *automated* and *easy* to use. It should be possible for the average user to create a tagged PDF with only minimal changes to their sources.

3 Some problems we faced

The Tagged PDF project has to redefine many LaTeX commands but this is in some sense an expected task. Before we describe the current state of the project we want to discuss some problems, outside of the Project Team's control, that we identified in the last year.

3.1 Tags are not visible

The screenshot in Figure 1 was made with Acrobat Pro, and that costs money. Free viewers and browser plug-ins normally don't show tags even if (as with Adobe Reader) they use them to improve, for example, the copy & paste heuristic. This missing visibility makes it difficult to convince users that tags are important. It also makes development in LaTeX difficult as package maintainers can't easily check the tagging. Fortunately, the situation is starting to change and some free PDF viewers like PDF-XChange and PDFix Desktop Lite now can be used to check the tags.

3.2 Missing PDF 2.0 support

The ISO spec for PDF 2.0 [5] is more than 6 years old. PDF 2.0 added various new features that are crucial for tagging. For example, it extends the tag set with tags like `Title`, `Aside` and `FEnote`. It also adds two new features, both needed for the tagging of math: 1) it declares the MathML namespace, and 2) it supports attaching embedded files to structure objects; then they are called "associated files". Such associated files can contain the source code or the MathML representation. With PDF 2.0 it is also possible to link to a structure instead of to a location on a page. LaTeX is able to produce PDF 2.0 and can make use of the new features.

But sadly PDF 2.0 also has a problem: it is not well supported by PDF viewers and consumers. Even Acrobat Pro is, at the time of writing, not able to handle tag namespaces and doesn't pass associated files attached to a structure through its accessibility API. PAC3 crashes if a PDF 2.0 files is loaded.

Structure Type	Children		Parents	
	Occ.	Structure Type	Occ.	Structure Type
P	0..n	NonStruct	0..n	Document
	0..n	Private	0..n	DocumentFragment
	0..n	Note	‡	Part
	0..n	Code	‡	Div
	0..n	Sub	0..n	Art
	0..n	Lbl	0..n	Sect
	0..n	Em	0..n	TOCI
	0..n	Strong	0..n	Aside

Figure 3: Part of the Parent-Child rules for the `P` tag

We have here a clear chicken-and-egg problem: As there are only few tagged PDF 2.0 files in the wild, no viewer correctly processes them, and because no viewer handles tagged PDF 2.0 files properly there is no incentive for applications to produce tagged PDF 2.0 files. However, there is some hope for changes: The PDF 2.0 reference is now available at no cost; ISO 32005, which handles the combined PDF 1.7 and PDF 2.0 tag set, has been released [6] and PDF/UA-2 will be released shortly too.

3.3 Parent-Child rules

Structure elements defined by the PDF spec can't be used freely: The PDF format defines containment rules and describes which standard tag is allowed as a child or parent of other elements. If you declare your own tag names you have to also declare a *role mapping* to the standard tags and your tags then have to obey the containment rules of the standard tags they map to. In PDF 1.7 the rules were only described in a rather vague way. PDF 2.0 introduced a large matrix for the PDF 2.0 tags and the new ISO 32005 [6] extends this matrix to combine the PDF 1.7 and 2.0 tags. Figure 3 shows an excerpt of the rules.

Until last year the `tagpdf` package didn't check such rules, so authors had to look up the lists manually to decide if a tagging structure is valid or not. With more automated tagging, this became problematic: it became too easy to create a structure that violates the rules, so a data structure to store and automatically check the parent-child relation has been implemented. This isn't trivial, as the checks have to take role mapping and slight differences between PDF 1.7 and PDF 2.0 into account, and handle special tags (`Part`, `Div` and `NonStruct`) that don't have their own rules but inherit the containment rule from their parent in the structures.

4 What is possible today: Tagging of "Leslie Lamport Documents"

The next milestone in the project is to enable the automated tagging of what we call "Leslie Lamport

Documents". These are documents which use environments and commands as described in the LaTeX manual [7]. They are based on the standard classes `article`, `report` and `book` and load only a restricted set of compatible packages.

The code is being developed in various modules in the `latex-lab` bundle, and can be loaded in the `\DocumentMetadata` command with the `testphase` key. The supported values for the key are described in `documentmetadata-support-doc.pdf`. In LaTeX 2023-06-01, `phase-III` is the most comprehensive value that loads almost all available modules, and also activates tagging.

```
\DocumentMetadata{testphase=phase-III}
\documentclass{article}
```

Please refer to the documentation for details of the different modules.

The following sections describe what is tagged in this phase. For engines, pdfLaTeX or LuaLaTeX are recommended; other engines may work but are not much tested and do not offer support for real space characters and so give less accessible PDFs.

4.1 Links

The `hyperref` package is fully supported and links (both internal and external) are automatically tagged.

4.2 Paragraphs

Paragraphs are tagged with the help of the para hooks. Page breaks in paragraphs are handled automatically [8] through patches of the output routine.

This mechanism usually works well, but there can be problems if paragraphs are directly nested (the parent-child rules do not allow that), if the para hooks are not balanced (e.g., if a low-level `\vbox` is not properly ended with a `\par`), or if a package operates with low-level paragraph commands such as `\everypar` in an unexpected way.

4.3 Footnotes

The `latex-lab` module for footnotes is a full reimplementation of the footnote code and adds hooks and configuration points needed both for tagging and for the configuration of footnote layouts.

This code is not compatible with classes like `memoir` or `KOMA`, or with packages such as `bigfoot`, `manyfoot` or `footnote` that redefine internals related to footnotes. A replacement for the `footmisc` package has been written, so this package is already supported.

4.4 Sectioning

The changes to sectioning commands are currently rather lightweight; only a few internal commands like `\@startsection` and `\@sect` are redefined. The tagging not only surrounds the titles with a structure but also adds a `Sect` structure around heading and body of every section (see figure 4). Classes like `memoir` or `KOMA` and packages like `titlesec` are incompatible.

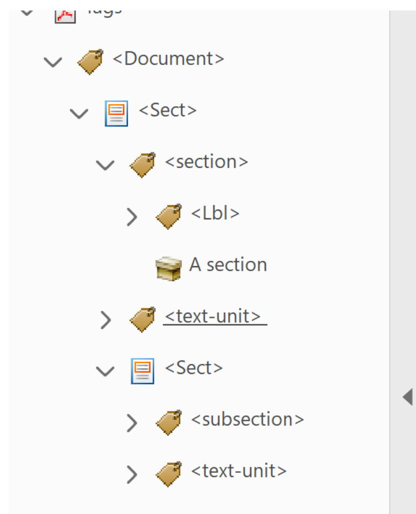


Figure 4: Tagging of sectioning commands

This implementation will not stay; the long-term plan is to reimplement the sectioning commands to offer more configuration options.

4.5 Tables of contents and similar lists

The implementation of the tagging support is rather lightweight and redefines, for example, `\@starttoc`, `\addcontentsline`, `\@dottedtocline`, `\l@part`, etc. Classes and packages like `memoir`, `KOMA`, `titletoc` and `etoc` are incompatible. Classes and documents that (re)define `\l@part`, `\l@chapter` or `\l@section` must adapt these definitions to add the hooks needed for tagging. An example of such a redefinition can be found in `ltugboat.cls`.

4.6 Display environments and lists

The list implementation in standard LaTeX serves a dual purpose: it implements true lists such as `itemize` and `enumerate`, and is also used as the basis for vertical blocks such as `center`, `quote`, `verbatim`, and for the theorem environments. These are all implemented as "trivial" lists with a single (hidden) item.

While this was convenient to get a consistent layout, it is not adequate when it comes to interpreting the structure of a document, because environments based on `trivlist` should not advertise themselves as being a "list" — after all, from a semantic point of view they aren't lists.

These environments have therefore been fully reimplemented and extended based on templates (from the `xtemplate` package). The code is currently incompatible with important packages like `enumitem`, `enumerate`, `fancyvrb`, `listings`, and also with theorem packages. Replacements or adaptations will be provided at a later stage.

4.7 Citations and bibliographies

Bibliographies are typically lists and so tagged by the list code of the previous section. Citations are tagged as `Reference` and point to the relevant bibliography entry. The `natbib` package is supported. `biblatex` is supported if `hyperref` is used.

4.8 Graphics

Graphics cannot be tagged automatically: For accessibility the author has to provide an alternative text. As an interface the new code adds[2] to `\includegraphics` and the `picture` environment an `alt` key that an author can use for this purpose. Ti*k*Z is not yet supported, but should follow soon.

```
\includegraphics
  [alt={This shows a duck}]{duck}

\begin{picture}
  [alt={This shows a duck too}](100,100)
   ...
\end{picture}
```

4.9 Floats

Floats (currently only `figure` and `table`) are tagged as “endfloats”. That means the structures are moved to the end of the structure tree.[3] An example tree is shown in figure 5. The code also changes the placement of the `hyperref` anchor: it is now moved to the beginning of the float. If a float contains two captions, the author has to manually mark how the float should be split into two float structures.

Also missing, at the moment, is support for the `float` package, `[H]` (“here”) floats (they will need a different structure), subcaptions and `\marginpar`.

4.10 `minipage` and `\parbox`

Initial support for `minipage` and `\parbox` has been added. But the content of such boxes can be arbitrarily complex, and it is unclear if the current code works as desired in all cases.

[2] The `alt` key was added to `\includegraphics` in LaTeX 2021-11-15 but does nothing if tagging is not used.

[3] Only the structures in the structure tree! This does not affect the typesetting!

Figure 5: Example structure tree with figure

4.11 Text commands

The LaTeX logo and `\emph` are tagged. To the logo an alternative text is added, and `\emph` is tagged as `EM`. `\textbf` is not tagged by default as it is too often used without a semantic meaning.

4.12 Math

There is also an early prototype for tagging formulas. It is not yet included in `phase-III` but can be loaded as an additional module:

```
\DocumentMetadata
  {testphase={phase-III,math}}
```

The code grabs the content of the math and reprocesses it. The code is compatible with `amsmath` (that package is explicitly loaded) but not compatible with packages defining their own math environments. Because of the missing PDF 2.0 support it is currently not possible to test if the provided tagging is in fact adequate.

The module can have side-effects on “fake math”, for example urls from the `url` package, which internally use math mode for technical reasons, and are tagged as `Formula` at the time of this writing. Such places must be found and handled separately.

4.13 Firstaid

`latex-lab` contains also a small module `firstaid` which contains small temporary corrections to external packages to improve their compatibility with the tagging code. Like the math module it must be loaded explicitly:

```
\DocumentMetadata
  {testphase={phase-III,firstaid}}
```

5 What is missing

We are not yet at the point where we can state that all elements described in Leslie Lamport’s manual on LaTeX are supported. One open point is `\savebox` and `\usebox` (and the TeX counterparts). Reusing boxes still needs some research to decide how to handle boxes with and without internal structure.

Indexes, glossaries and similar lists are currently not properly handled either. They will not generate errors, but an index created with the standard `theindex` environment will tag every item and subitem and subsubitem as a paragraph of its own, so important semantic information about the nesting is lost.

And last, there is the structure with which this article started: tables. It is not very difficult to surround table rows and cells with `TR` and `TD` structures. But a properly tagged table should surround header cells with `TH` and complex tables need to reference the header from the data cells. The standard LaTeX input doesn't mark up the headers of the table and it is not yet clear how the syntax can be extended to detect headers reliably.

This was one of the three topics in the "tagging workshop" we ran a day before the 2023 TUG conference in Bonn, Germany. A report from this workshop is given elsewhere in this issue [1].

6 Summary

Starting with LaTeX 2023-06-01, quite a large set of standard documents can be tagged in an automated way — as an example this (*TUGboat*) article itself has been tagged with the code.

There is still much to do. We are grateful for everyone who tests the code and gives feedback. For this we have opened a dedicated Github repository for the tagging project where issues can be reported and discussions can be opened:

`https://github.com/latex3/tagging-project`

References

[1] D. Carlisle, U. Fischer, F. Mittelbach. Report on the LaTeX Tagged PDF Workshop, TUG 2023. *TUGboat* 44(2):267–269, 2023. `doi.org/10.47397/tb/44-2/carlisle-taggedpdfworkshop23`

[2] U. Fischer. *The* `tagpdf` *package*. `ctan.org/pkg/tagpdf`

[3] U. Fischer. Creating accessible pdfs with LaTeX. *TUGboat* 41(1):26–28, 2020. `tug.org/TUGboat/tb41-1/tb127fischer-accessible.pdf`

[4] U. Fischer. On the road to Tagged PDF: About StructElem, marked content, PDF/A and squeezed bärs. *TUGboat* 42(2):170–173, 2021. `doi.org/10.47397/tb/42-2/tb131fischer-tagpdf`

[5] ISO. *ISO 32000-2:2020(en)*, 2nd ed., 2020. PDF 2.0. `iso.org/en/contents/data/standard/07/58/75839.html`

[6] ISO. *ISO/TS 32005:2023*, 1st ed., 2023. PDF 1.7 and 2.0 structure namespace inclusion. `iso.org/en/contents/data/standard/04/58/45878.html`

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

[8] F. Mittelbach. Taming the beast — advances in paragraph tagging with pdfTeX and XeTeX (2021): Automatic paragraph tagging with the pdfTeX and XeTeX engine[s] now possible. `latex-project.org/news/2022/09/06/TUG-online-talks-21-22/`

[9] F. Mittelbach, the LaTeX Project Team. Quo vadis LaTeX(3) Team — a look back and at the upcoming years. *TUGboat* 41(2):201–207, 2020. `tug.org/TUGboat/tb41-2/tb128mitt-quovadis.pdf`

◇ Ulrike Fischer
LaTeX project team
Bonn
Germany
`ulrike.fischer (at) latex-project.org`

◇ Frank Mittelbach
LaTeX project team
Mainz
Germany
`frank.mittelbach (at) latex-project.org`
ORCID 0000-0001-6318-1230

Report on the LaTeX Tagged PDF workshop, TUG 2023

David Carlisle, Ulrike Fischer,
Frank Mittelbach

Contents

Introduction

On the afternoon before the formal conference program, the LaTeX project held a workshop, led by Ulrike Fischer, on generating tagged PDF from LaTeX. The workshop was well attended with more than thirty people participating — a good mix of package developers and end users. We thank DANTE e.V. for very generous financial support.

The workshop was split into three parts. Firstly, a general introduction to tagging in PDF. Secondly, a demonstration of the process that a class or package maintainer should take to modify the code to produce well-tagged PDF. The `acmart` class was used for the example as its author, Boris Veytsman, was attending the workshop. Finally, we had a more open discussion on issues and desired syntax for structured tables.

1 Tagged PDF

Ulrike gave a brief overview of how PDF tagging encodes the structure of a document in a PDF file and why with is important not "just" for accessibility requirements. Readers are encouraged to read the introduction to the `tagpdf` package for a similar, more detailed exposition.

1.1 Engine considerations

The recommended engine for tagging is luaLaTeX. For legacy documents pdfLaTeX is supported, but it has some small problems, for example, it sometimes doesn't insert real space characters in places where they are needed. It also requires more compilations to build the correct tagging structure.

Other workflows such XeLaTeX or LaTeX–`dvips` are not recommended as real space characters can't be inserted in these cases. In order for accessibility tools to distinguish inter-word spaces from inter-letter kerns and other spacing adjustments, words need to be separated by space characters (U+0020) even if the spacing is further adjusted. It is not feasible to add these space characters just using the macro layer, and currently only pdfLaTeX and luaLaTeX have engine-level support to add them.

When developing or updating packages or classes for tagging, one always needs to test tagging with at least pdfLaTeX *and* luaLaTeX. They use different methods to create the basic MC-chunks (called "marked-content sequences" in PDF reference manuals) and create the structure tree (namely, PDF literals in pdfTeX and luaTeX node attributes in luaTeX).

1.2 Tagging commands and Tagging activation

The `tagpdf-base` package provides dummy versions of all important tagging commands. It is loaded automatically by `\DocumentMetadata` but can also be loaded by other packages.

The `tagpdf` package is loaded (and tagging is then automatically activated) by using a `phase` key:

```
\DocumentMetadata{testphase=phase-III}
```

It is possible to produce untagged PDFs with the new code in `latex-lab` (both for the final version or in draft mode to speed up compilation), but a suitable interface is currently missing. It will be added later.

1.3 Compatibility with older formats and legacy code

The `latex-lab` code and tagging in general require a current LaTeX or even `latex-dev`. If necessary, a class or package can test the format version with `\IfFormatAtLeastTF` or `\@ifl@t@r\fmtversion` to provide fallbacks for older formats.

Whether a document uses `\DocumentMetadata` can be tested with `\IfDocumentMetadataTF`. With `\tag_if_active:TF` it is possible to test if tagging is active.

At the moment tagging can only be activated through the testphase keys in `\DocumentMetadata`, which cannot be used in a class because this command must come before `\documentclass`. This means that at this point in time a class cannot trigger tagging — during the test phase this is the decision of

doi.org/10.47397/tb/44-2/tb137carlisle-taggedpdfworkshop23

the author of the document. A class can only check the state and issue an error or a warning if tagging is not activated.

1.4 Tools

To check the tagging structure you need a tool, or several. Some possible options:

- Adobe Acrobat Pro (non-free, `adobe.com`)
- PDF XChange Editor (a free version is available, `pdf-xchange.eu`)
- PDFix (free Desktop Lite version is enough, `www.pdfix.net`)
- PAC 2021 (a checker, `pdfua.foundation/de/pac-2021-der-kostenlose-pdf-accessibility-checker`). Currently it doesn't work with PDF 2.0.
- Big Faceless PDF Library (free trial version, `bfo.com/download`). A Java library that can also be used to dump information into text files.
- RUPS (`itextpdf.com/products/rups`)
- An online service you can try: `ngpdf.com`

2 Updating a LaTeX class

In the second part of the workshop the general principles of how to update a package to be compatible with PDF tagging were discussed, illustrated with specific examples from `acmart` class.

2.1 Adapting packages and classes

Broadly, packages and classes have to handle two problems:

- They often redefine internal LaTeX commands and environments (directly or through a package or class they load) and this breaks the tagging support provided by `latex-lab`.
- Any new commands and environments defined by the package need to be adjusted to add or correct the tagging.

Some general recommendations on strategies to address these issues:

- For all existing redefinitions of LaTeX's core commands, check their purpose and consider if the redefinition can be avoided, e.g., by making using of the recently-introduced hooks in various core commands of LaTeX for precisely this reason, or by making a feature request to enhance the LaTeX command to support your use case directly or via new hooks.
- New commands and environments built on top of existing commands can inherit the tagging support from the kernel. For example, bibliographies are typically simply lists. Check if the resulting structure is ok.
- Paragraphs are automatically tagged through the paragraph hooks. Not every structure is allowed inside a paragraph. It is therefore important to check how your own commands behave both in horizontal and vertical modes.
- Complex commands can be difficult to tag correctly. For a first draft it is possible to put a minimal structure around them and then to stop tagging. As an example, a complicated `\maketitle` command could be handled as follows, i.e., by disabling tagging for the title:

```
\AddToHook{cmd/maketitle/before}
  {\tagstructbegin{tag=Title}%
   \tagmcbegin{}\tagstop }
\AddToHook{cmd/maketitle/after}
  {\tagstart\tagmcend\tagstructend}
```

Participants were, and readers of this report are, encouraged to give feedback. The tagging support is still in development. If something doesn't work as expected or is too complicated we need to know about it. Comments can be added as issues or discussions at `github.com/latex3/tagging-project`.

2.2 `acmart`

The `acmart` class is used to typeset articles for the Association for Computing Machinery. It is around 3500 lines of code, based on `amsart`, and supports various journal styles.

The aim of the workshop was not to fully update the entire class to be tagging-aware, but to show the steps needed for this, and to show how relatively simple changes can already significantly improve the automatic tagging in documents, even when generated by large production journal code.

2.2.1 Review of packages and commands

The initial step was to generate a list of all packages used by the class, around 40 packages in this case. Some are known not to affect tagging, some (notably those affecting footnotes such as `manyfoot` and `nccfoots`) may affect footnote tagging. They would need to be checked in a final version; specifically, whether newer versions of the packages are tagging-aware or whether there is "first-aid" support for the package in the `latex-lab` code.

The main issue addressed in this session was classes that copy "original" definitions of standard LaTeX commands, but often they copy older versions and so are missing updates, in particular the (relatively) recent changes to add LaTeX hooks and tagging support.

For example, in the case of `acmart`, it intends to simply restore standard LaTeX section headings (undoing the changes made by the underlying `amsart` class). It does this by redefining `\@startsection` and related commands using copies from an older LaTeX format. This has the effect of undoing the additions to support tagging section headings added by the tagging code.

In this case, a simple fix would be to modify the class to save the definitions (for example with `\DeclareCommandCopy`) before loading `amsart` and restore them afterwards. This has the effect of restoring the tagging-aware section commands, if they were activated by `\DocumentMetadata`.

Pointers were given to changes that would be required in other parts of the class. Tables of contents would need changes equivalent to the changes made for tagging to the standard `\@dottedtocline` and `\@starttoc` commands. The class has quite extensive code modifying footnotes and this must be checked for tagging. Perhaps in initial versions, tagging would be disabled for footnotes.

As noted above, the title page handling (as is common in journal classes) is rather complicated and would need some custom tagging support, but can in initial versions be easily customized not to tag, so that no invalid tag structures are generated.

Similarly, the class has redefinitions of `minipage` that would conflict with minipage tagging and need to be checked.

In the workshop there was only time to look at the redefinitions of standard commands made by the class, and how to recover tagging support based on the support implemented for the standard definitions. What was not addressed in the session — but would be needed for a fully tagging aware class file — would be to add appropriate tagging support to any new commands and environments defined by the class. This requires understanding of not only the TeX coding details but also, and more importantly, an understanding of the intended structure implied by the commands; that is, what tagged structure would be desirable in each case.

Going forward, this raises an important point for class and package maintainers. To produce well-tagged documents it is not only important to produce a pleasing visual layout for new constructs, it is equally important to think about what structures are represented, and how that structure should be tagged. For example, "misusing" other elements for purely visual effects is likely to produce completely inadequate structural results.

3 Tables

The final part of the workshop involved a lively discussion of markup for tables. It is already possible to use the low-level tagging commands from `tagpdf` to tag tables and individual cells, although the current test phase does not have code to automatically infer table structure.

The main problem is that classic LaTeX `tabular` markup does not distinguish table headers or any other structural features, it is purely concerned with the visual layout.

This leads to two related issues:

1. What would be a good table markup for new documents?
2. What heuristics could or should be used to infer table structure for the existing corpus of LaTeX documents?

For the first issue, markup used by LaTeX table packages were considered, also ConTeXt table markup, and the table markup in HTML (which is very close to the final required structured tagging in PDF).

For the second issue, various possibilities were discussed ranging from a view that it is better never to "guess" and that no table headings should be inferred, through a simple heuristic such as always assuming the first row is a heading, to more detailed heuristics looking at fonts and/or position of `\toprule` from `booktabs` or `\endhead` from `longtable`.

This is still very much an area of ongoing activity for the LaTeX team, and people are invited to contribute to the discussion which is now online in the GitHub discussion site set up for the tagging project mentioned earlier: `github.com/latex3/tagging-project/discussions/1` has some thoughts from workshop participants on the table syntax. Feel free to join the party with further ideas.

◇ David Carlisle
LaTeX project team
Oxford, UK
david.carlisle (at) latex-project.org

◇ Ulrike Fischer
LaTeX project team
Bonn, Germany
ulrike.fischer (at) latex-project.org

◇ Frank Mittelbach
LaTeX project team
Mainz, Germany
frank.mittelbach (at) latex-project.org

Enhancing accessibility of structured information via ‘Tagged PDF’

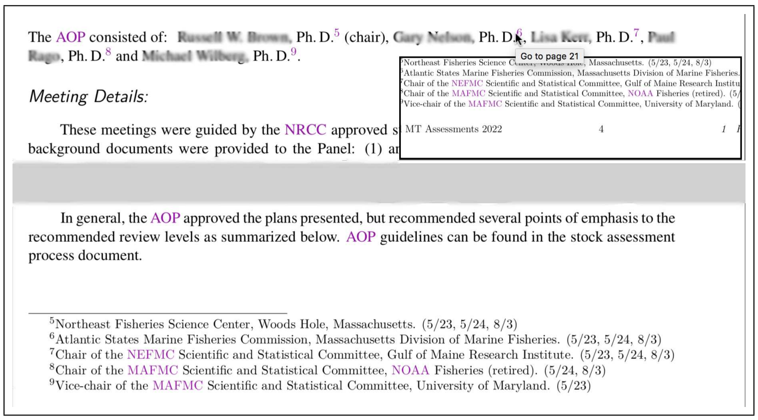
Ross Moore

It is common practice in LaTeX to use coding such as in the following listing, resulting in a layout as shown in Figure 1. (Names are smeared for privacy.)

```
The \AOP\ consisted of:
<-1st member->\footnote{<-affil->} (chair),
<-2nd member->\footnote{<-affil->},
<-3rd member->\footnote{<-affil->}, ...
```

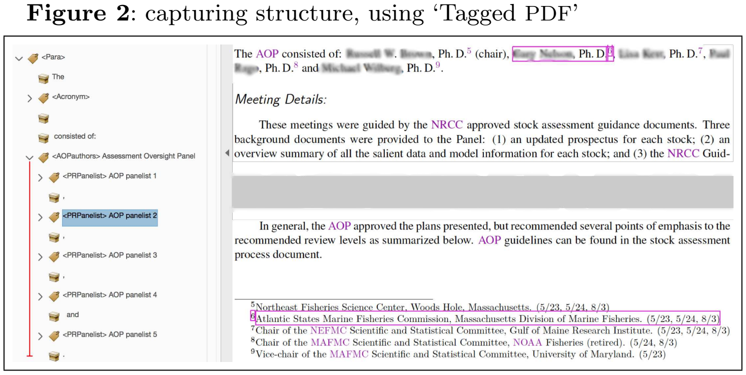
When reading the names one ‘glances’ down the page to check the affiliation, or in some PDF readers, hovers over the footnote marker link to induce the pop-up, as shown in Figure 1.

Figure 1: sequence of names/authors and affiliations



However, what if ‘glancing’ is not possible? For example, the reader is visually-impaired, for whom a pop-up also is not appropriate. Even the concept of ‘page’ need not be relevant or useful.

Figure 2: capturing structure, using ‘Tagged PDF’



In several countries there are now anti-discrimination laws or policies[1] which for the most part embody the principle that “... agencies must give disabled employees and members of the public access to information comparable to the access available to others.”

To achieve a result, as in Figure 2, the first step is to rearrange coding as in this next listing.

```
The \AOP\ consisted of:
\begin{AOPpanel}
\AOPchair{<-1st ...->\footnote{<-affil->}}
\AOPmember{<-2nd ...->\footnote{<-affil->}}
 ...
\AOPlastmember{<-5th ...->\footnote{<-affil->}}
\end{AOPpanel}
```

using definitions in a preamble or package, such as:

```
\newenvironment{AOPpanel}{\ignorespaces}{}
\newcommand{\AOPchair}[1]{\PRPpanel{#1} (chair)}
\newcommand{\AOPmember}[1]{, \PRPpanel{#1}}
\newcommand{\AOPlastmember}[1]{ and \PRPpanel{#1}.}
\newcommand{\PRPpanel}[1]{#1}
```

One can see on the right in Figure 2 that the visual layout is the same as in Figure 1. However tagging in the left panel of Figure 2 shows how the punctuation is separated out from the panelist information; with a separate structure element for each. Each use of a macro `\AOPchair`, ..., corresponds to a detail in how the information fits into the single paragraph. `\PRPpanel` handles the real information, combining both the name with the affiliation in a footnote; Figure 3 shows this in greater detail. This is all enclosed in a structure for the `{AOPpanel}` as a whole.

The way the user-defined macros and environment are used, provides a way to capture the ‘semantics’ of this sequence of panelist names and affiliations.

The reason for using a separate `\PRPpanel` macro, rather than one named `\AOP...` is because there are other similar structures in this same document. Differences in the tagging, and hence the semantics being captured, can be associated with the surrounding environment, whilst `\PRPpanel` can be common to all such situations. In Figure 3 we see part of an earlier page in the same document. The footnote numbers are earlier, and there is no existing paragraph surrounding the `{PRPauthors}` structure.

```
\newenvironment{PRPauthors}%
 {\ignorespaces}{\bigskip\bigskip}
\newcommand{\PRPchair}[1]{\PRPpanel{#1} (chair)}
\newcommand{\PRPauthor}[1]{, \PRPpanel{#1}}
\newcommand{\PRPlastauthor}[1]{ and \PRPpanel{#1}}
```

[1] https://www.section508.gov/manage/laws-and-policies/

doi.org/10.47397/tb/44-2/tb137moore-pdf-html

Figure 3: tagging of the PRP authors

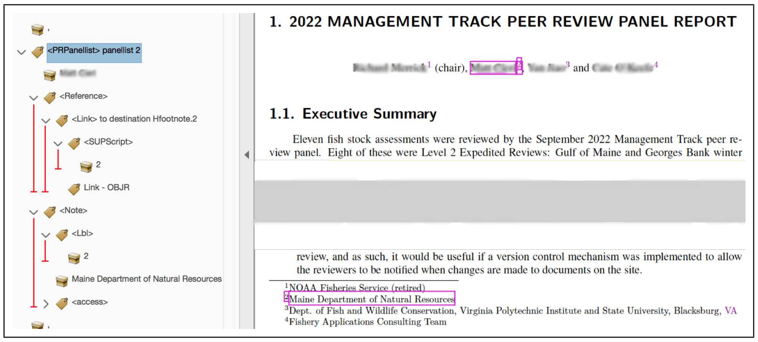


Figure 3 shows the result from `\PRPpanel`, as tagging a `<PRPpanelist>` structure, numbered in sequence. Firstly the name is placed, followed by a `<Reference>` structure, consisting of the superscripted `\footnotemark` as anchor for an active hyperlink. This is followed immediately by the `<Note>` structure coming from the `\footnotetext`, repeating the mark as a `<Lbl>` then placing the text.

For a person reading with Assistive Technology (AT) navigating via structure, the analog of 'glancing down the page' is to simply move by a single element from the `<Reference>` to its following `<Note>`, or to use the hyperlink to get to the same location.

While the first `\footnotemark` uses a custom `<SupScript>` structure, which is 'Role-mapped' to a standard `<Span>`, the second usage does not. There the `<Lbl>` structure has an attribute dictionary:

```
<< /O /Layout  /TextPosition /Sup >>
```

For deriving to HTML (see below) this attribute cannot be applied to the `<Link>` structure; hence the extra `<SupScript>` is used.

Constructing a 'Tagged PDF' document which is structured as shown in Figures 2 and 3 requires a highly sophisticated LaTeX macro system to produce the tagging of both structure and content. The **`tagpdf`** LaTeX package is under development, but not yet ready for real-world documents such as the one shown here. Instead this author and colleagues use coding developed by the author, under the name `tpdf`. A full description of `tpdf` is beyond the scope of this article.

However, it should be clear that part of any such system must include extending or re-defining the expansions of the macros used for user input; that is, the `\AOP...` and `\PRP...` macros, the environments `{AOPpanel}` and `{PRPauthors}` as well as LaTeX internal commands (e.g., for processing footnotes and hyperlinks) and environments generally. With environments there are 'hooks' which allow for additional coding to be executed in 4 different places via `\AddToHook{`⟨hook⟩`}`; so `env/AOPpanel/before` and `env/AOPpanel/after` allow coding to be added before and after the environment itself is processed. These can be used to control how the environment's structure fits into that of the surrounding document. Similarly, hooks `env/AOPpanel/begin` and `env/AOPpanel/end` can place more coding *inside* the environment, affecting how information is handled within the environment itself, without affecting outside. Both pairs of hooks are useful.

Furthermore, one can modify `\AOPpanel` and `\endAOPpanel` from the expansions provided when the environment was set up using `\newenvironment`. There is a `\patchcmd` macro in the **`etoolbox`** package, but often it is easier to redefine the expansions directly, whilst keeping a copy of the original expansion in case needed within the redefined version.

```
\NewCommandCopy\LTX@AOPpanel\AOPpanel
\newcommand\TPDF@AOPpanel[1]{....}
\NewCommandCopy\AOPpanel\TPDF@AOPpanel
```

Use similar coding for adjusting `\endAOPpanel`.

For example, the `\TPDF@PRPpanel` replacement for `\PRPpanel` must implement the following tasks:

- close off any preceding text, leaving the `<Para>` open;
- increment the counter for the kind of panelist;
- start the next `<PRPpanelist>` structure, numbered appropriately;
- process the argument `#1`; that is, place the name as tagged text, then allow a `\footnote{...}` command to be processed with its argument.

It is a modified version of internal commands called by `\footnote` that handles tagging of the footnote markers and text, resulting in the tagging seen in Figure 3. After that is all done, structure is closed to the level of the original `<Para>`, to allow any further panelists to be included, as seen in Figure 2.

PDF reuse as HTML and XML, attributes and classes

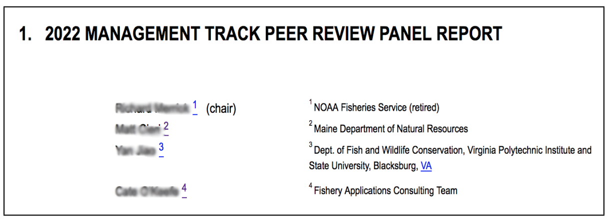
Typically visually impaired people prefer to read documents via HTML, using Assistive Technology (AT) such as screen-readers, Braille keyboards, and more. Previously PDF files have been regarded as inaccessible, compared with HTML. Standards such as PDF/UA are intended to redress this situation; but old habits and preferences cannot be expected to be altered until AT software updates sufficiently to better handle Tagged PDFs built to conform to newer standards.

Figure 4: HTML coding derived from `\PRPpanel` usage

```
<span> </span>
<div class="PRPanelist" data-pdf-se-type-original="PRPanelist" data-pdf-se-type="Div" id="PRPanelist.6">
(-- 2nd author --)<a href="#Note.15" class="Xlink" data-pdf-se-type="Link"
aria-details="Note.15" aria-label="AOP panelist 2 affiliation" id="Link.0994">
<sup data-pdf-se-type-original="SUPScript" data-pdf-se-type="Span" id="SUPScript.06">6</sup></a>
<p class="PRPaffil" data-pdf-se-type="Note" name="Hfootnote.6" id="Note.15">
<sup data-pdf-se-type="Lbl" id="Lbl.242"><span>6 </span></sup>
 Atlantic States Marine Fisheries Commission, Massachusetts Division of Marine Fisheries.  (5/23, 5/24, 8/3)
<span class="Access" data-pdf-se-type-original="access" data-pdf-se-type="Span" id="access.242"><span>
</span></span></p></div>
```

Fortunately there is good online software[2] that 'derives' a well-structured single HTML file from a valid PDF/UA document. Figure 5 shows the result of using this with our real-world example PDF, for a sequence of panelists, viewed in the Firefox browser. Being a single HTML file, there is no concept of 'page' to determine where to place footnotes. Instead, the `<Note>` structure is floated to the right using CSS rules, explained below.

Figure 5: HTML version of the PRP authors



To better understand Figure 5, one needs to look at part of the raw HTML coding, as shown above in Figure 4. One sees that each HTML tag (`<div`, `<sup`, `<p`, `<span`) has a unique `id="..."` attribute, inheriting the corresponding structure's unique name from the PDF. Namely, the paragraph with `id="Note.15"`, which comes from the `<Note>` structure having the footnote text as can be seen in Figure 1, becomes the target for the hyperlink from `<a href="#Note.15"`.

Also shown, as lighter colored attributes and values, are the PDF structure-element names, with `-original` indicating a 'Custom' structure name. This maps to a 'Standard' name as shown in the attribute which follows. Such user-defined HTML attributes would be ignored by web browsers, unless specially set up to handle them.

For layout on the HTML page, the important attributes are `class="..."`. The specified names and affiliations are subject to CSS style rules:

```
.PRPanelist { clear:right; }
.PRPaffil { float:right; font-size:small;
  margin:0 0 0 5px; width:60%; }
```

[2] ngPDF: `https://ngpdf.com/`

This use of `float` and `clear` is what turns an otherwise horizontal sequence into what appears to be a 2-column vertical listing.

For Accessibility, having anchor text being just a single digit, say '2', is not very informative. Hence the `aria-label="..."` tells what kind of information is found at the link target. A longer description, via `aria-details="..."`, is at the target location itself. With such attributes for all internal hyperlinks, WCAG Level A Success Criterion 2.4.4[3] is fulfilled for this document. This uses the concept of 'Accessible Name',[4] here built from the anchor-text and aria-label value.

Figure 6: link attributes

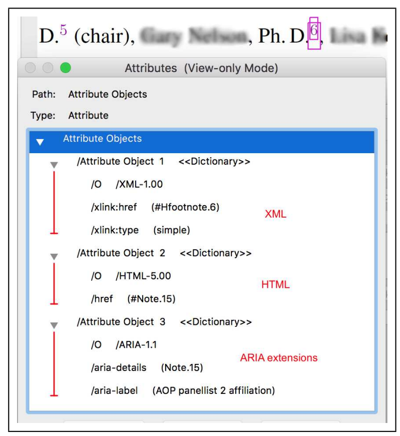


Figure 6 shows the array of attribute dictionaries specified for the hyperlink within the PDF document, as shown in figures 2 and 5. These control how the link works when exported into either HTML or XML formats. Notice how the XML version uses a target of `Hfootnote.6`, which is the name of the PDF 'Destination'. On the other hand, HTML requires the 'Structure Destination' of `Note.15`.

Within the PDF, the hyperlink is implemented as an 'Annotation' of subtype 'Link', as shown in

[3] Web Content Accessibility Guidelines: `https://www.w3.org/TR/WCAG21/#link-purpose-in-context`

[4] `https://www.w3.org/WAI/ARIA/apg/practices/names-and-descriptions/`

Figure 7: Link annotation dictionary

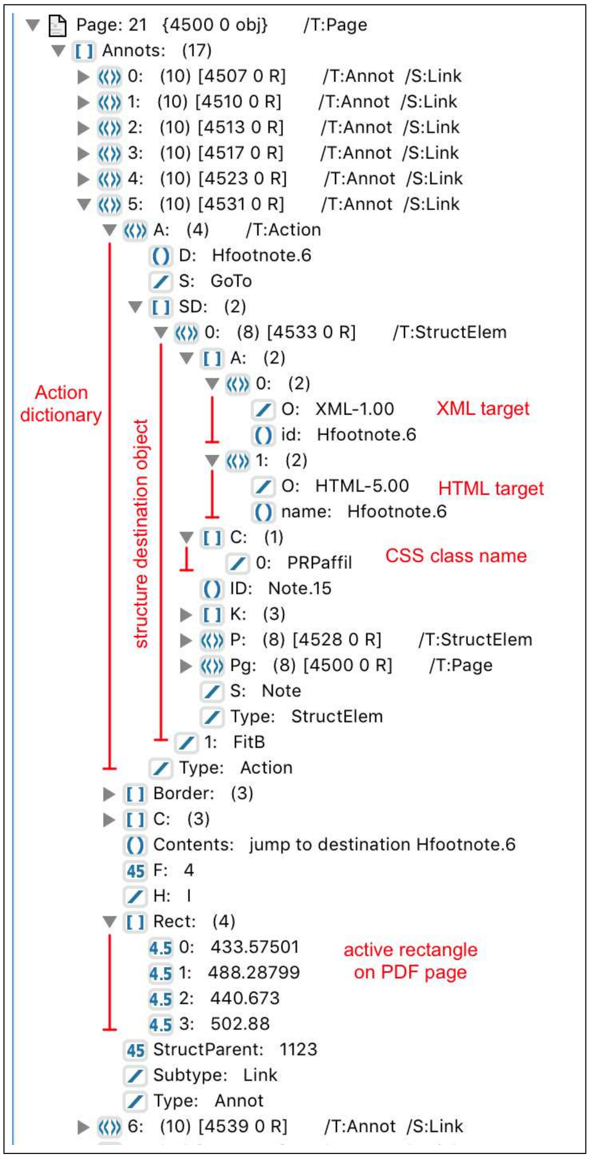
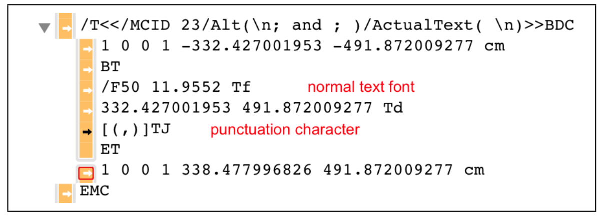


Figure 7. This dictionary includes a clickable 'Rectangle' on the specific page, as well as specifying the 'Action' to be taken when clicked. The `D` field value of `Hfootnote.6` names the target within the PDF, while `SD` provides a reference to the structure, having object number 4533 and named as `Note.15`, found at that location; that is, a 'Structure Destination'. Notice how this object also provides its name for export to both XML and HTML formats, specifying the respective 'Attribute' type (either `id` or `name`). Also there is the 'Class' name of `PRPaffil`, used for CSS styling as discussed earlier.

One further detail, seen in Figure 5, is that the punctuation has been suppressed to become simply `<span> </span>`. This is achieved in the PDF content stream as shown in Figure 8. Appearing as a comma within the PDF using the 'show string' of `[(,)]TJ`, the `/Alt(...)` and `/ActualText(...)` provide alternatives that can be spoken by a screen-reader or used with text extraction.

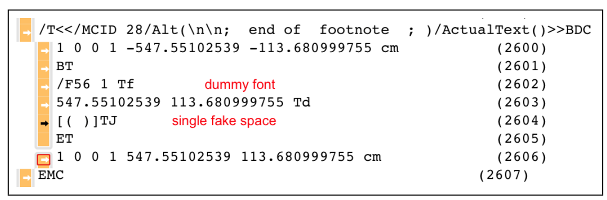
Figure 8: handling variants for punctuation



Access tags

A special kind of structure and content is provided by so-called 'Access' tags, whose presence can be seen by the `<access>` in Figure 3, and near the bottom of the HTML listing in Figure 5. These use `/Alt` and `/ActualText` similarly to the above, having a 'show string' of `[( )]TJ` (see Figure 9) which places a space character, but using the 'dummy font' having width 0.001 pt, as obtained with the `\pdffakespace` primitive. This space is imperceptible within the typeset PDF, yet is still selectable. With an empty string for `/ActualText` it adds no content to the HTML and XML exports.

Figure 9: 'Access Tag' after footnote



However, a non-empty `/Alt(...)` string, as in Figure 9, affects what is spoken by some (but not all) screen-reading software, including Adobe Reader's 'Read Out Loud' facility. With ' ; ' creating a slight pause, the descriptive markup conveyed via these 'Access' tags is separated from actual content being read. Furthermore, the complete stream that is read can be exported as 'Text (Accessible)' from Adobe software. In Figure 10 we see how the sequence of names shown in Figures 1 and 2 will be spoken or exported. The judicious use of `\n\n` inserts line-breaks to help break up the text stream nicely upon export.

Conclusions

Here we have attempted to show, using a 'real-world' example document, how with a relatively simple

Figure 10: Accessible Text for screen-reading

```
The A O P consisted of: ; start of AOPauthors block ;  (-- 1st author --) , Ph.D.
; refer to footnote 5 ;
; start of footnote 5 ; Northeast Fisheries Science Center, Woods Hole, Massachusetts. (5/23, 5/24, 8/3)
; end of footnote ; (chair)
; and ; (-- 2nd author --), Ph.D.
; refer to footnote 6 ;
; start of footnote 6 ; Atlantic States Marine Fisheries Commission, Massachusetts Division of Marine
 Fisheries. (5/23, 5/24, 8/3)
; end of footnote ;
; and ; (-- 3rd author --), Ph.D.
 ...
```

adjustment of source coding, the semantics of structured information can be captured explicitly within a 'Tagged PDF' document. Each use of a user-defined macro or environment provides a place where extra meaning can be captured for presentation, not just visually on a page, but preserved for export into other formats. The various figures show how and where this information can be stored inside the PDF, and also where it then appears within derived HTML and Text-only export formats.

By giving consideration also to CSS styling for the exported HTML, and WCAG recommendations for enhancing Accessibility, one can indeed provide "disabled ... access ... comparable to ... others". Even the tiniest details, such as punctuation, can be handled so as to enhance each alternative view, without compromising the high-quality visual layout that is (LA)TEX's hallmark.

Technical notes

All the images and figures used in this article originate from a 'real-world' document intended for publication and release into the public domain.[5] Researchers' names have been deliberately blurred, so as not to be easily extractable. Similarly, names have been replaced by generic place-holder strings in the textual outputs of Figures 5 and 10. Those authors and panel members have nothing whatsoever to do with the techniques of 'Tagged PDF' production and the export into other formats being discussed here.

Figure 1 is a (doctored) view using Apple's 'Preview' application, using 'TEXshop'[6] on a MacBook; most of the page's content was removed and the rest compacted to better display the relevant parts. Figures 2, 3 and 6–9 are views using Adobe's 'Acrobat Pro' application,[7] involving various panels to display visual content, 'Tags' tree, an 'Attribute' array, an internal view of a portion of the 'Structure Tree', and internal parts of a 'Page Content' stream in raw PDF. All graphic editing of screen-shots was done using 'GraphicConverter'[8] for macOS.

Figure 4 is a screenshot, taken while using the 'Firefox' browser, of the HTML webpage derived by 'ngPDF' at the website[2] stated earlier. Figure 5 is from a text-editor application ('Erbele'[9] for macOS) which was used to work with the HTML document source. With Figure 10 Apple's 'TextEdit'[10] was used to present the Text (Accessible), exported from 'Acrobat Pro'; but any text-editing software would be sufficient for this. Other textual code-listings were done using either 'Erbele' or the default editor for 'TEXshop', which was the application for composing this article. It was also the application used to build the "real-world" 185-page 'Tagged PDF' document that has supplied the examples, built using LATEX. All software used was running on an Apple 'MacBook Pro'[11] device.

◇ Ross Moore
School of Mathematical and Physical Sciences
Macquarie University
Sydney, Australia
ross.moore (at) mq dot edu dot au
https://researchers.mq.edu.au/en/persons/ross-moore

[5] at http://www.science.mq.edu.au/~ross/TaggedPDF/FallMT2022/ as FallMT2022.html and FallMT2022.pdf.

[6] TEXshop: https://pages.uoregon.edu/koch/texshop/

[7] Acrobat Pro: https://www.adobe.com/products/acrobat-pro-cc.html

[8] GraphicConverter: https://www.lemkesoft.de/en/products/graphicconverter/

[9] Erbele text editor: https://apps.apple.com/de/app/erbele/id1595456360

[10] TextEdit: https://support.apple.com/en-au/guide/textedit/welcome/mac

[11] MacBook Pro: https://www.apple.com/macbook-pro/

An HTML/CSS schema for TeX primitives — generating high-quality responsive HTML from generic TeX

Dennis Müller

This paper uses sTeX3. The semantically annotated HTML version of this paper is available at `url.mathhub.info/tug23css`.

Abstract

I present a schema for translating TeX primitives to HTML/CSS. This translation can serve as a basis for (very) low-level TeX-to-HTML converters, and is in fact used by the RusTeX system — a (somewhat experimental) implementation of a TeX engine in Rust, used to convert LaTeX documents to HTML — for that purpose.

Notably, the schema is accurate enough to yield surprisingly decent (and surprisingly often "the exactly right") results on surprisingly many "high-level" LaTeX macros, which makes it adequate to use in lieu of (and often even instead of) dedicated support for macros and packages.

1 Introduction

Translating LaTeX documents (partially or fully) to HTML is a difficult problem, primarily because the two document formats address very different needs: TeX is intended to produce statically laid out documents with fixed dimensions, ultimately representing ink on paper. HTML, on the other hand, assumes a variety of differently sized and scaled screens and consequently prefers to express layouts in more abstract terms, the typesetting of which are ultimately left to the browser to interpret, ideally responsively — i.e. we want the document layout to adapt to different screen sizes, ranging from 8K desktop monitors to cell phone screens.

This means that there is no one "correct" way to convert TeX to HTML — rather there are many choices to be made; most notably, which aspects of the static layout with fixed dimensions described by TeX code to preserve, and which to discard in favour of leaving them up to the rendering engine, thus explaining the plurality of existing converters.

Naturally, many LaTeX macros are somewhat aligned with tags in HTML; for example, sectioning macros (`\chapter`, `\section`, etc.) correspond to `<h1>`, `<h2>`, etc.; the `{itemize}` and `{enumerate}` environments and the `\item` macro correspond to `<ul>`, `<ol>` and `<li>`, respectively; and so on. Most converters therefore opt for the reasonable strategy of mapping common LaTeX macros directly to their closest HTML relatives, with no or minimal usage of (simple) CSS, effectively focusing on preserving the *document semantics* of the used constructs (e.g. "paragraph", "section heading", "unordered list"). In many situations, this is the natural approach to pursue, especially if we can reasonably assume that the document sources to be converted are sufficiently "uniform", so that we can provide a similarly uniform CSS style sheet to style them, and this is largely the way existing converters work. To name just a few:

- LaTeXML [7] focuses strongly on the *semantics*, using XML as the primary output format and heuristically determining an author's intended semantics of everything from text paragraphs (definitions, examples, theorems, etc.) down to the meaning of individual symbols in mathematical formulae; achieving great success with `ar5iv.org`, hosting HTML documents generated from TeX sources available on `arxiv.org`.
- TeX4ht [12] focuses on plain HTML as output with minimal styling, going as far as to (optionally) replace the `\LaTeX` macro by the plain ASCII string "`LaTeX`".
- Pandoc [5] largely focuses on the most important macros and environments with analogues in all of its supported document format to convert between any two of them, e.g. TeX, Markdown, HTML, or `docx`.
- Mathjax [6] focuses exclusively on macros for mathematical formulae and symbols, allowing to use TeX syntax in HTML documents directly, which are subsequently replaced via JavaScript by the intended presentation.

However, the approach described above has notable drawbacks: Firstly, it requires special treatment of LaTeX macros that plain TeX would expand into primitives, and the number of LaTeX macros is virtually unlimited — CTAN has (currently) a collection of 6399 packages, tendency growing, which get updated regularly, and authors can add their own macros at any point. Supporting only the former is a never-ending task, and providing direct HTML translations for the latter is impossible. This is made worse by the very real and ubiquitous practice among LaTeX users of copy-pasting and reusing various macro definitions and preambles assembled from stackoverflow, friends and colleagues, and handed down for (by now *literally*) generations, even in situations where (unbeknownst to them) "official" packages with better solutions (possibly supported by HTML converters) exist.

For example, I myself have happily reused the following macro definition for years:

doi.org/10.47397/tb/44-2/tb137mueller-primitives

```
\usepackage{amsmath,amssymb}
\def\forkindep{\mathrel{\raise0.2ex\hbox
  {\ooalign{\hidewidth$\vert$\hidewidth
  \cr\raise-0.9ex\hbox{$\smile$}}}}}
```

... neither knowing nor caring what it actually does other than that it allows me to typeset $A \mathrel{\raise0.2ex\hbox{\ooalign{\hidewidth$\vert$\hidewidth\cr\raise-0.9ex\hbox{$\smile$}}}}_C B$ ("A and B are *forking-independent* (or *non-forking*) over C"; a concept in model theory)[1] — despite there being a Unicode symbol (0x2ADD) and a corresponding LaTeX macro `\forksnot` in the `unicode-math` package. If we want to maximise coverage, we therefore need a reasonable strategy for arbitrarily elaborate unexpected LaTeX macros.

Secondly, by generating rather plain HTML, we guarantee that the resulting presentation is *neutral* and can be easily adapted by users via their own CSS stylesheets — the "morally correct" thing to do. However, it also severely clashes with the expectations of (casual) users that the result look roughly the same as the PDF. After all, LaTeX documents are written by authors in a way that is inevitably optimized for a particular layout and arrangement of document elements. Subsequently discarding them in favor of as-plain-as-possible HTML that optimizes more for the "document semantics" of the components than their (precise) optics yields plain looking HTML that is immediately perceived as ugly, "not what I want" and requires lots of massaging to achieve a similar aesthetic level as the PDF generated by pdfLaTeX. And *aesthetics matter* — that's why TeX was created in the first place.

Thirdly, by focusing on supporting as many LaTeX macros as possible directly, conversion engines tend to neglect support for primitives in multiple senses of "support" — indeed, I found it difficult to find any existing TeX documents of mine that "survive" any of the existing HTML converters for a realistic comparison, typically dying with no output or only initial, badly formatted fragments.

The RusTeX system is a TeX-to-HTML converter born out of our needs in the sTeX project [4, 10]. The `stex` package allows for annotating LaTeX documents (in particular mathematical formulae and statements) with their (flexi-)formal semantics. These documents are subsequently converted to HTML, preserving both the (informal) document layouts as well as the semantic annotations in such a way that knowledge management services acting on the semantics can be subsequently integrated via JavaScript. Our existing corpora of sTeX documents cover a wide range, from individual fragments (definitions, theorem statements, remarks, ...) up to research papers, lecture slides in `beamer`, and book-like lecture notes that usually *include* the slides between text fragments, all of them using a multitude of (typical and untypical, official and custom) packages, preambles and stylings.

We consequently want to translate the sources for all these heterogeneous documents to HTML such that 1) the results look as similar to their PDF counterparts as possible, 2) the semantic annotations are preserved as XML attributes, and 3) (most importantly) conversion succeeds for any error-free document, regardless of packages and macros used, so that at least the semantic annotations can be extracted, even if the presentation is occasionally (somewhat) broken.

Contribution Motivated by the above, this paper describes RusTeX's rather extremal point in the design space of LaTeX-to-HTML converters: The goal is to mimic the core TeX expansion mechanism (i.e. pdfLaTeX) as closely as possible and map the resulting sequence of TeX primitives to (primarily) `<div>`s with CSS attributes, while avoiding the neverending amount of work required for the special treatment for non-primitive TeX macros. Ideally, this allows for achieving full error-free coverage with respect to converting full documents, and yielding HTML that looks reasonably close to what a user would expect.

Of course, if we *only* care about aesthetics, we might as well render the generated PDF in the browser directly. So as an addendum to the above, we should add the desideratum that the HTML remain "reasonably recognizable as HTML": for example, plain text in paragraphs (or horizontal boxes) should actually be represented as plain text in the resulting HTML — in fact, we want to leave to the browser as much as possible of what a browser does best: break lines in paragraphs, size boxes based on their contents (where we want them to be), and arrange components based on available (screen) space, according to constraints imposed by our CSS schema.

RusTeX's git repository [9] contains a `.tex` file with test cases for (and beyond) all the following, and the HTML generated from them for direct comparison. Additionally it contains the PDF and HTML produced from my Ph.D. dissertation [8], which serves as a particularly good test case for several reasons:

1. I was a typical LaTeX user when I wrote it, with no particular knowledge of TeX's internal workings, and hence unbiased by what I would nowadays do to avoid problems.

[1] Possibly sourced from `tex.stackexchange.com/questions/42093/what-is-the-latex-symbol-for-forking-independent-model-theory` — I needed and found it some time around 2013.

2. I spent a lot of effort on making it look nice by the usual means — copy-pasting from elsewhere and using whatever package google tells me to use to achieve the desired effect.
3. It is a 215 page document using everything from elaborate formulas, syntax-highlighted code listings, various figures and tables, and color-coded environments (using `tcolorboxes`) for remarks, theorems, examples, definitions, etc.

The HTML generated from our sTeX corpora can be found at `url.mathhub.info/stex`, including this paper (see link above), which thus additionally serves as a demonstration of the examples below (notably, with two column mode deactivated). They also power our *course portal* at `courses.voll-ki.fau.de`, where students at our university can access semantically annotated course materials and various didactic services generated from them. The full CSS schema is also available.[2]

Disclaimer I am not arguing to eschew dedicated support for LaTeX and package macros entirely — document semantics can be important, for example for accessibility reasons. Additionally, while the translation presented here is surprisingly effective, it has clear limitations, especially on the scale of *individual characters* (see section 9).

Hence, the ideas of this paper should be seen as a reasonable *fallback* strategy usable *in conjunction with* dedicated support for macros. Indeed, RusTeX too currently implements (a few) package macros, namely `\url`, `\not` and `\cancel`, `\underbrace` and `\overbrace`, `\marginpar`, the `{wrapfig}` environment, and (somewhat embarrassingly) `\LaTeX`.

In fact, if this paper has a purpose beyond reporting on what I consider to be an interesting experiment, it should be the following: *Taking TeX primitives seriously pays off aesthetically*, can save a lot of work and effort, and where possible, I encourage developers of TeX-to-HTML converters to take them seriously *in addition to* dedicated support for macros.

Furthermore, many of the techniques described below are the result of more-or-less informed experimentation; in many cases, better ways to represent TeX primitives in HTML might exist. I appreciate feedback and suggestions for improvements.

2 General architecture

As mentioned, RusTeX attempts to mimic the behaviour of pdfLaTeX as closely as possible. As such, it implements the behaviour of the primitive commands available in plain TeX, ε-TeX and pdfTeX, amounting to 293 + 47 commands, excluding primitive "register-like" commands such as `\everyhbox`, `\baselineskip`, `\linepenalty`, etc. Their precise behaviour has been determined from (obviously) the bible [3] and the manuals for ε-TeX and pdfTeX, but also often reverse engineered via extensive experimentation.

The program starts by locating, via `kpsewhich`, a user's `pdftexconfig.tex` and `latex.ltx` and processing them first. This requires that the user have a LaTeX distribution set up, but subsequently makes sure that RusTeX behaves as close to the local LaTeX setup as possible.

Tokens are expanded in the expected manner down to the primitives, which cause state changes, impact expansion, or ultimately end up fully processed in RusTeX's *stomach* waiting to be output as HTML. The latter primitives are the subject of this paper.

`pgf` (and thus `tikz`) is handled via an adapted version of the existing SVG driver and thus omitted here. Images are inserted directly in the HTML in base64 encoding.

In lieu of a *shipout routine*, box registers for *floats* (as well as `\inserts` such as footnotes) are occasionally heuristically inspected and inserted, but this mechanism is due for a more adequate treatment and hence also omitted.

2.1 Trees and fonts

Naturally, HTML is a tree structure of nested nodes. Somewhat counter-intuitively, so are TeX's stomach elements, but unfortunately at the cost of attaching information such as the current font, font size, color, etc., directly to the individual "character boxes". If we wanted to introduce a `<span>` node for every individual character, we could mimic this directly in HTML — however, this approach is too extreme even for my taste. Luckily, in almost every situation where colors and fonts are changed, the changes are achieved via LaTeX macros that align with TeX's "stomach tree". For example,

```
\textbf{\textcolor{blue}{some} \emph{text}}
```

clearly entails a tree of font and color changes, which ideally should be represented as a corresponding HTML tree:

```
<span style="font-weight:bold">
  <span style="text-color:blue">some</span>
  <span style="font-style:italic">text</span>
</span>
```

[2] `github.com/slatex/RusTeX/blob/master/rustex/src/resources/html.css`

And indeed, all three macros (`\textbf`, `\textcolor`, `\emph`) introduce TeX groups for their arguments, assuring that these changes reflect a tree structure.

Consequently, RusTeX can (somewhat) safely add special nodes to the stomach on font changes, changes to the color stack, or *links* (as produced by `\pdfstartlink`). As these are (usually) local to the current TeX group, the stomach consequently also keeps track of when TeX groups are opened and closed. If such changes (i.e. their start and end points) conflict with other stomach elements' delimiters, such as boxes or paragraphs, they are appropriately closed and subsequently reopened, e.g.:

```
Some paragraph \begingroup \itshape
this is italic \par
New paragraph, still italic \endgroup not
italic anymore
```

typesets as:

Some paragraph *this is italic*
New paragraph, still italic not italic anymore

and would yield HTML similar to:

```
<div class="paragraph">
  Some paragraph
  <span style="font-style:italic">
    this is italic
  </span>
</div>
<div class="paragraph">
  <span style="font-style:italic">
    New paragraph, still italic
  </span>
  not italic anymore
</div>
```

In general, the nodes produced by font changes and similar commands are considered "*annotations*": If these nodes have no children, or a single child that modifies the same CSS property, they are discarded or replaced by their only child. If they have a single child or are the only child of their parent node, the corresponding `style`-attribute is attached to the relevant node directly. Only in the remaining case is an actual `<span>` node produced in the output HTML.

To deal with fonts in general, it should be noted that most TeX fonts are freely available in a web-compatible format (e.g. `otf`) online; we *could* consequently use the actual fonts used by TeX in the output PDF. In practice, we prefer to have adequate Unicode characters in the HTML output, rather than ASCII characters representing a position in a font table. Consequently, RusTeX instead hardcodes fonts as pairs of 1) a map from ASCII codes to Unicode strings and, 2) a sequence of font modifiers (e.g. *bold*, *italic*). The former is used to produce actual characters, the latter to choose appropriate CSS attributes as above.

Currently, RusTeX fixes Latin Modern as the font family used, but somewhat nonsensically obtains font metrics the same way as TeX, by processing the `tfm` files on demand [2], providing only rough approximations of the actual values (in HTML).

2.2 Global document setup

At `\begin{document}`, RusTeX determines 1) the current font and its size, 2) the page width (as determined by `\pdfpagewidth`) and 3) the text width (as determined by `\hsize`), and attaches them as corresponding CSS attributes to the `<body>` node — the page width determining the `max-width` and the calculation (⟨*page width*⟩−⟨*text width*⟩)/2 determining the `padding-left` and `padding-right` properties. The latter is important to accommodate e.g. `\marginpar` and related mechanisms, and is discussed more precisely in section 5.

3 Boxes and dimensions

Clearly, the most important primitives to get "right" are (horizontal or vertical) *boxes*, produced by `\hbox`, `\vbox` and variants (`\vtop`, `\vcenter`), as they are the primary means that more elaborate macros use to achieve their aims. They also serve as good examples of the complexities involved when translating to HTML.

Boxes have five important numerical values that matter with respect to how they are typeset: `width`, `height`, `depth`, `spread` and `to`, which we will discuss shortly.

Horizontal boxes (as produced by `\hbox`) — as the name suggests — have their contents arranged horizontally, and vertical ones vertically. This is nicely analogous to the CSS *flex model*, so naturally, we can associate boxes with CSS flex display values. An entire document can be thought of as a single top-level vertical box. Hence:

```
.hbox, .vbox, .body {
  display: inline-flex;
}
.vbox, .body { flex-direction: column; }
.hbox { flex-direction: row; }
```

An important distinction that matters here is that between the actual *contents* of the box and its *boundary*. Usually, the dimensions of a box are computed from the dimensions of its children — which, conveniently, is analogous to HTML/CSS, so in the typical case we do not need to bother with them at all and leave those up to the rendering engine:

```
.hbox, .vbox, .body {
  width: min-content;
  height: min-content;
}
```

Whenever possible, we *avoid* precisely assigning dimensional values in HTML and defer to the ones computed by the rendering engine. This is important to account for discrepancies between HTML and TeX, e.g. regarding the precise heights of characters, lines, paragraphs, etc.

However, the dimensions of a box can be changed after the fact, using the `\wd`, `\ht` and `\dp` commands (corresponding to `width`, `height` and `depth`, respectively). If these dimensions are changed, the *contents* and how they are laid out are not changed at all, but the typesetting algorithm, when putting "ink to paper", will proceed *as if* the box had the provided dimensions. This allows macros to layer boxes *on top* of each other; in the (very common) most extreme case by making boxes take up no space at all. For example:

```
\setbox\myregister\hbox{some content}
\wd\myregister=0pt \ht\myregister=0pt
\dp\myregister=0pt
\box\myregister other content
```

This will produce a horizontal box with the content "some content" with all dimensions being 0 from the point of view of the output algorithm, meaning the "other content" following the box will be put directly *on top* of the box, like so:

some content / other content

Hence, we *do* have to occasionally consider the actual (computed or assigned) dimensions of TeX boxes and other elements.

Regarding boxes, we attach actual values for `width`/`height` to their HTML nodes *if and only if* they have been *assigned* fixed values, and let

```
.hbox, .vbox { overflow: visible; }
```

We can then achieve the same effect in HTML via:

```
<div class="hbox" style="width:0;height:0;">
  some content
</div> other content
```

3.1 `width`/`height` vs. `to`

Things get more interesting if the assigned values for the dimensions of the box are *larger* than the actual box contents — this tells us how we need to align the contents of boxes vertically and horizontally. This, however, is also where the `to`-value of a box comes into play:

Setting `\wd`=$\langle$*val*$\rangle$, for any $\langle$*val*$\rangle$, for a horizontal box, as mentioned, does not impact the way the box *content* is laid out. However, using `\hbox to=`$\langle$*to-val*$\rangle$`{...}` *does*, while also setting the `width` of the box: The `to`-attribute instructs TeX to arrange the contents of the box "in line with" the box being $\langle$*to-val*$\rangle$ wide. For example:

```
\hbox{some box content}
\hbox to \textwidth{some box content}
```

some box content		
some	box	content

This example is deceptive in that it suggests the box contents were evenly spread out across the $\langle$*to-val*$\rangle$ of the box, but this is not so. Consider:

```
\hbox to \textwidth{
  \hbox{some}\hbox{box}\hbox{content}
}
\hbox to \textwidth{%
  \hbox{some}\hbox{box}\hbox{content}%
}
```

someboxcontent (centered)
someboxcontent

It's not that the individual content elements in the box are spread out evenly; instead, they are left-aligned and glue items (e.g. from space, newline, and tab characters) behave approximately as if they were `\hfil` — i.e. they take up as much space as they can in the containing `\hbox`. And while subsequently the box has a width of $\langle$*to-val*$\rangle$, it can be changed with `\wd` like any other box:

```
\setbox\myregister\hbox to \textwidth{%
  \hbox{some}\hbox{box}\hbox{content}%
}\wd\myregister=0pt \box\myregister
\setbox\myregister\hbox to \textwidth{%
  \it some box content%
}\wd\myregister=0pt \box\myregister
```

someboxcontent / *some*	*box*	*content*

This distinction between the three values `width`, `to`, and "total width of the box's children" forces us to distinguish between a) the box itself (i.e. its contents) with its (potential) `to` value, and b) its "boundary box", i.e. subsequently assigned `width`s and `height`s. The same holds analogously for the `to` value and `height` of a vertical box:

```
.hbox { text-align: left; }
.vbox { justify-content: flex-start; }
.hbox-container, .vbox-container {}
```

where the `.hbox-container`-class is used for *assigned* `width`s and `height`s, and `to` translates to the width of the `.hbox` itself. Making spaces behave as they should in an `\hbox` forces us to style them accordingly:

```
.space-in-hbox {
  display: inline-block;
  margin-left: auto;
  margin-right: auto;
}
```

Using this class for spaces (directly) in `\hbox`es makes the remaining content stretched across the full width of the box, as in the examples above.

Notably, TeX allows for *negative* values in dimensions, which CSS does not. To capture the resulting behaviour, whenever a dimension (e.g. `width`) is < 0, we set the `width` CSS property to 0, and attach (in this case) `margin-right:`⟨*width*⟩ to the HTML node (analogously `margin-top` for `height`).

Finally, the `spread` parameter can be used *instead* of `to` and *adds* the provided dimension to the computed width/height of the box; e.g. if `\hbox{foo}` has width 15pt, then `\hbox spread 15pt{foo}` has width $15 + 15 = 30$pt:

> Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque.foo   Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque.

Annoyingly, the only way to accommodate this seems to be to compute the "original" value, add the `spread` value, and attach that as the final width/height to the <div> node.

3.2 Depth and rules

So far, we have considered `width` and `height`, but TeX has an additional dimension for boxes that CSS does not: `depth`, which measures the extent to which a given box extends *below* the baseline of the parent box. Depth is rarely important, or rather, matters primarily when manipulating individual characters, which CSS is currently not capable of for reasons explained later. However, notable, and not uncommon, exceptions are explicitly *assigned* depth values, in particular for `\vtop` boxes.

To better understand depth, let's turn our attention to the `\vrule` primitive, which produces a colored box of the provided dimensions:[3]

```
Lorem ...
\vrule width 10pt height 10pt depth 10pt
Lorem ...
```

[3] `\hrule` is implemented analogously, except for using `display:block` instead of `inline-block`.

> Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque. █ Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque.

This creates a black box with 10pt width and a total 20pt height, centered at the *baseline* of the current line: extending 10pt *above* the baseline (the `height`) and 10pt *below* (the `depth`).[4]

Such a box with the right dimensions can be easily produced using CSS:

```
.vrule {
  display: inline-block;
}
```

The individual <div>s are then provided `background`, `width` and `height` (=`height`+`depth`) properties corresponding to the color and the dimensions of the `\vrule` — in the above example:[5]

```
style="background:#000000;height:20pt"
```

The tricky part is ensuring that the box is correctly positioned with respect to the surrounding text (or other elements). As above, the solution is to wrap the `.vrule` <div> in a `.vrule-container` with the same height as the inner <div>, and adding `margin-bottom:-`⟨*depth*⟩ to the inner `.vrule`. This not only allows for moving the box the specified amount below the baseline, but also makes sure that the "boundary" that the rendering engine computes for positioning elements has the relevant dimensions as well.

If a rule has no explicitly provided width/height, TeX gives it a thickness of 0.4pt, and other dimensions fitting the current box:

```
\hbox{ \vrule
 \vbox{ \hbox{some} \hrule \hbox{text}}
\vrule }
```

> | some / text |

We can easily set the width of the `\hrule` with `width:100%` to achieve the same effect. Unfortunately, the same does not work with `\vrule` and its height in HTML, as an artifact of when and how the heights of boxes are computed by the rendering engine. In those situations, we have to distinguish between paragraphs and `\hbox`es: In the former case we heuristically set the height to the current font size; in the latter (since we are in a flex box), we

[4] Note the gap between the second and third line of text, caused by the depth of the `\vrule`.

[5] For simplicity's sake, we will use the same dimensions (in `pt`) in both TeX code and CSS; in practice, we scale `1pt` in TeX to a value in `px` units.

can set `align-self:stretch` to make the rule fit the containing box.

3.3 `\vbox` vs. `\vtop` vs. `\vcenter`

`\vtop` behaves like `\vbox`, except that where a `\vbox` is vertically aligned at the *bottom* of the parent box's baseline, a `\vtop` is vertically aligned at the top with the surrounding text, extending downwards. `\vcenter` is vertically aligned at the center and is only allowed in math mode:

```
some text \vbox{\hbox{some}\hbox{vbox}}
text \vtop{\hbox{some}\hbox{vtop}}
text $\vcenter{\hbox{some}\hbox{vcenter}}$
text
```

some text some vbox text some vtop text some vcenter text

Internally, the three types of vertical boxes differ precisely in their a priori `depth`s and `height`s. As long as these are not subsequently reassigned (using `\ht` and `\dp`), we can achieve the same effect much more accurately by using the `vertical-align` property, that covers the same primary *intent* of the three types of vertical boxes:

```
.vbox{ vertical-align: bottom }
.vtop{ vertical-align: baseline }
.vcenter{ vertical-align: middle }
```

We now need to be careful with changing the *height* of a `\vtop` box, however: Since the primary vertical dimension of a `\vtop` corresponds to its *depth* (below the baseline), *increasing* its height actually corresponds to moving the box contents upwards *without changing the amount of space* it takes up *below* the baseline:[6]

```
Lorem ...
 \setbox\myregister\vtop{\hbox{some}\hbox{vtop}}
\ht\myregister=20pt\box\myregister
Lorem ...
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque. some vtop Lorem ipsum dolor sit amet, consectetur adipiscing elit pellentesque.

This can be approximated in HTML by setting both the `margin-top` *and* `bottom` CSS properties of the `.vbox-container` to the value ⟨*height*⟩−⟨*current line height*⟩: The `bottom` property moves the box upwards, while the `margin-top` property makes sure that the boundary box grows accordingly, instead of the moved box overlapping with other elements.[7]

Conversely, if we manipulate the *depth* of a `\vtop`, we can set the `height` of the `.vtop` HTML node itself to ⟨*depth*⟩+⟨*current line height*⟩.

Annoyingly, it turns out that height/depth manipulations on `\vbox`es and `\vtop`s (respectively) do not play well with `vertical-align` CSS properties within paragraphs — the boxes are not correctly aligned vertically. When explicitly setting these dimensions, it is therefore necessary to, as with `\vrule`, introduce an intermediate HTML node with class `.vbox-height-container` to achieve the effect.

4 Paragraphs

At a first glance, paragraphs in TeX seem largely straightforward:

```
.paragraph {
  text-align:justify;
  display: inline-block;
  margin-top: auto;
}
```

The `margin-top:auto` assures that paragraphs are vertically aligned at the bottom of `\vbox`es.

Any horizontal material (e.g. text, `\noindent`, `\unhbox`) outside of a paragraph, or an `\hbox` (and similar constructions) *opens* a new paragraph, and `\par` closes it again.

If we were primarily interested in document semantics without caring about the page layout dictated by TeX, we could be done at this point. However, in TeX, paragraphs have fixed widths dictated by several parameters and commands, including `\hsize`, `\leftskip`, `\rightskip`, `\hangindent` and `\hangafter`, and `\parshape`. This matters when a paragraph is opened inside a `\vbox`. Consider, for instance

```
Lorem ipsum \vbox{Some Text} Lorem ipsum
```

Lorem ipsum Some Text Lorem ipsum

The `Some Text` in the `\vbox` opens a new paragraph, including indentation, and that paragraph has width `\hsize`, regardless of its contents. The `\vbox` itself then inherits the full width of the containing paragraph.[8]

Approximating this behaviour (in the absence of dedicated macro support) matters, for example to

[6] Again, note how the three lines in the paragraph are pushed apart by the unchanged depth and new height of the box.

[7] The same idea is used for `\raise`/`\lower`.

[8] Here, we've set `\hsize` to a smaller value to attempt to demonstrate the effect without breaking the layout of this article too much.

accommodate {minipage}s, tcolorbox and similar packages. This is also one case where TEX is significantly more flexible than HTML/CSS: \hangindent and \parshape do not have CSS equivalents. While in principle it might be possible to "emulate" them using empty <div> nodes with float attributes, we currently ignore them and proceed as if the whole paragraph were typeset according to the rules applying to the last line; e.g. the last entry in the \parshape list.

The relevant parameters can subsequently be condensed into three attributes, in the simplest case computed thusly: 1) the actual width of the text (\hspace−(\leftskip+\rightskip)), and 2) left and right margins (\leftskip and \rightskip), which we translate to the CSS attributes min-width, margin-left and margin-right, respectively.

Notably, to accommodate macros that make use of computed dimensions of various boxes, we need to approximate TEX's line breaking algorithm to make sure that the computed heights of paragraphs are reasonably accurate.

5 Responsiveness and relative widths

The above suggests that we need to hardcode the absolute widths of both the document as a whole (in the sense of \textwidth/\pagewidth) as well as the widths of paragraphs and \hboxes. This is of course undesirable in that it destroys responsive layout in HTML. Ideally, we would prefer to use *relative* widths in terms of percentages.

Regarding the document width, this is easily resolved: Instead of letting width:⟨*text width*⟩, we set max-width:⟨*text width*⟩. This way, the page accommodates smaller screens, but if enough screen space is available will default to the size for which the document was originally designed.

Relative widths in general however only work as expected if the direct parent of a node has a fixed assigned width, and as previously mentioned, insofar as possible we want to defer the precise dimensions of HTML nodes to the rendering engine. Moreover, once we have a box with width:0, no percentage will get us back to a non-zero value. Both problems would be solvable if CSS allowed for inheriting attribute values from arbitrary ancestors, but since it does not, we need to be more creative:

Instead of directly inheriting, we can use a *custom* CSS property --current-width and initialize it as --current-width:min(100vw,⟨*text width*⟩); width:var(--current-width) in the body. (The vw unit is 1% of the width of the "viewport", i.e., browser windows.) This achieves the same effect as the more naive approach above, but now allows for stating other widths in the body of the HTML node as values relative to the --current-width attribute.

Using this approach, all relative widths in a document are now relative to the *current document's* initial \textwidth. This is problematic in the context of sTEX, where the \inputref macro largely replaces TEX's \input: Besides allowing for referencing source files relative to a math archive (i.e. a "library" of document snippets), which is important for building modular libraries, when converting to HTML \inputref simply inserts a reference to the file, which can subsequently be dynamically inserted into the referencing document. This obviates both the need to reprocess the same file for every context in which it occurs, as well as to rebuild all referencing files every time any of the \inputrefed files change. Notably, such \inputrefs often occur deeply nested, e.g. a file with a short individual definition might be \inputrefed in an {itemize} environment in a definition block in a framed beamer slide within lecture notes.

This entails that we would like to inherit widths from *the closest ancestor with a fixed assigned width* > 0 (e.g. the innermost \item in the example above) rather than the <body>, and update the value of --current-width accordingly, to accommodate any document context in which the HTML node might (dynamically) occur.

For example, given a top-level \vbox with width 0.5\textwidth (e.g. a {minipage}), we would like to do:

```
<div class="vbox" style="--current-width:calc(
  0.5 * var(--current-width));
  width:var(--current-width)">...</div>
```

Unfortunately, CSS does not allow for self-referential attribute updates; so we have to use an intermediary custom attribute --temp-width and an inner <span> to do the following:

```
<div class="vbox" style="--temp-width:calc(
    0.5 * var(--current-width));
    width:var(--temp-width)">
  <span style="display:contents;
    --current-width:var(--temp-width)
  ">...</span></div>
```

to achieve the desired effect. While this is ugly from an implementation point of view, it allows for variable viewport widths and solves the problem with inheriting widths *through* boxes of size 0.

6 Skips and text alignment

In section 4, \leftskip and \rightskip were considered as simple dimensions, but skips have three components: A base dimension, an (optional) *stretch* factor, and an (optional) *shrink* factor. A skip represents a (horizontal or vertical) space that is ideally ⟨*base dimension*⟩ wide/high, but can stretch or shrink according to the other two components to fit the current page layout. Stretch and shrink factors have one of four units `pt` (or any fixed unit), `fil`, `fill` or `filll`, the latter three representing "increasingly infinite" stretch/shrink factors.

Skips are used to introduce vertical or horizontal space, using (most commonly) the `\hskip` and `\vskip` commands. Focusing solely on their base dimensions for now, both can be represented as empty <div> nodes with corresponding `margin-left` or `margin-bottom` values, respectively. Conveniently, this works with both positive and negative base dimensions, and we can use the same mechanism for `\kern`, which for all practical purposes behaves like `\hskip` or `\vskip` with zero stretch/shrink. This allows us to cover both of the following cases:

```
\noindent some text \hskip20pt some text\par
\noindent some text \hskip-20pt some text
```

some text   some text
some textsome text

If we add a stretch factor, we can produce the following:

```
\noindent some text \hskip20pt plus 1filll
some text\par
```

some text   some text

Unfortunately, CSS has no analogue for stretch and shrink factors. For *shrink*, this largely causes no serious issues. *Stretch* factors however are primarily used to achieve (primarily horizontal) *alignment*. Left-aligned, centered, or right-aligned content is achieved in TeX by inserting corresponding skips; so the best we can do is to represent skips as the CSS `text-align` property:

If `\leftskip` or `\rightskip` have stretch factors, we compare them and set the alignment for the paragraph accordingly. For `\hbox`, we need to inspect the contents of the box for initial and terminal occurrences of relevant skips, compare them, and derive the intended alignment depending on which is "bigger".

Additionally, we can add `margin-left:auto` to the <div>s corresponding to skips iff they have a stretch factor of (at least) `1fil`; however, this only works in `\hbox`es (not in paragraphs), and does not necessarily behave correctly in conjunction with other skips.

Thankfully, text alignment seems to be the primary regularly occurring situation where skips are noticeable and important to represent accurately in the HTML, which this heuristic approach seems to cover reasonably well. While discrepancies between PDF and HTML can be easily found, they are usually not severe.

7 Math mode

For stomach elements in math mode, we naturally use Presentation MathML. Translating the relevant primitives to MathML is largely straightforward and covered elsewhere [11], with the slight "modernization" that we prefer CSS over MathML attributes. Since the font used for MathML depends on the rendering engine, and some of these are rather unsatisfactory (e.g. vanilla Firefox under Ubuntu), we can explicitly set the font to `Latin Modern Math` for a more unified look. Skips and kerns are implemented as above, but using <mspace> nodes instead of <div>.

Regarding font sizes, we can either defer to the rendering engine or use the sizes from TeX — in which case we need to make sure that we override the CSS rules imposed by the rendering engine via:

```
msub > :nth-child(2), msup > :nth-child(2),
mfrac > * , mover > :not(:first-child),
munder > :not(:first-child) {font-size:inherit}
```

More pressingly however, occurrences of `\hbox` or `\vbox` in math mode require us to "escape" back to HTML in <math> elements. While not officially supported, using <mtext> nodes for that works well in both Firefox and Chromium (and with some hacking with MathJax). However, when doing so, various CSS properties are inherited from the default stylesheet for MathML. Hence, whenever we escape back to horizontal or vertical mode, we explicitly insert the parameters of the current text font, and set:

```
mtext {
  letter-spacing: initial;
  word-spacing: initial;
  display: inline-flex;
}
```

As mentioned in [11], spacing around operators (i.e. <mo> nodes) is governed by an operator dictionary. The spacing rules are in principle well-chosen and best left to the rendering engine. TeX can change these however, using the commands `\mathop`, `\mathbin`, etc. To accommodate this functionality, we can explicitly set left and right padding based on TeX's math character class, and set:

```
mo {padding-left: 0;padding-right: 0}
```

Notably, this works (as of May 2023) in Firefox, but not in Chromium-based browsers,[9] where the spacing determined by the operator dictionaries is effectively a *minimum* that cannot be reduced further.

Changing these spacing factors can occasionally be important when composing symbols from more primitive ones. For example, the \Longrightarrow macro $\Longrightarrow$ concatenates the symbols $=$ and $\Rightarrow$ with a negative \kern between them — in which case unintended spacing between the two symbols can break the intended result.

8 \halign

The \halign command is the primitive which most LaTeX commands and packages use to lay out *tables*, and not surprisingly, its closest correspondent in HTML is <table> nodes. However, as with text alignment, effects that in HTML are achieved via attributes of the parent node (<table>, <tr> or <td>) are achieved in TeX via content elements *in* the individual cells — or between them: Where a table in HTML is exactly a sequence of rows consisting of cells, in TeX, the \noalign command allows for inserting vertical material *between* rows, which is used to insert horizontal lines (e.g. \hline) or determine the spacing between rows. Borders and spacing between cells are achieved via \vrules and skips.

Hence, we have to face two major problems when translating \haligns to <table>s:

1. If we want to accommodate spacing, text alignments and borders, we need to "parse" the contents of cells and \noalign blocks to determine which CSS attributes to attach to the <table>, <tr> and <td> nodes. This is made worse by the fact that the margin attributes on <td> and <tr> nodes have no actual effect.
2. The *height* of a <tr> is computed from the *actual* height of its children, and even enclosing a whole cell in a <div> with height:0 does not change the actual height of the relevant <tr>.

While the former problem is inconvenient but solvable, the latter becomes severe when we consider some less obvious situations for which \halign is used: For example, the \forkindep macro mentioned above uses \ooalign to combine the two characters $|$ and $\smile$, which in turn uses an \halign to superimpose them, forcing us to make the rows narrower than <tr>s allow for.

Therefore we use the CSS grid model for \halign rather than the (seemingly more adequate) <table>:

```
.halign {
  display:inline-grid;
  width: fit-content;
  grid-auto-rows: auto;
}
```

with cells being styled like .hbox with the additional attributes height:100%;width:100%, and any \halign with n columns being given the additional CSS attribute grid-template-columns:repeat(n, 1fr). This aligns the individual cells almost exactly like <table> would, but gives us the more control over their intended heights.

\noalign vertical material can now be inserted in a .vbox <div> with grid-column:span n. This entirely obviates the need to implement special rules for visible borders or spacing between rows/columns: The existing treatment for \vrule/\hrule and skips produces (almost universally) the desired output out of the box.

Notably, empty cells in \halign are not actually empty. Consider:

```
\halign{#&#\cr a&b\cr c&d\cr&\cr e&f\cr}
```

ab
cd

ef

The third row has no content, but we still get a row that has roughly the same height as the other three. We can remedy this effect via:

```
\baselineskip=0pt\relax
\halign{#&#\cr a&b\cr c&d\cr&\cr e&f\cr}
```

ab
cd
ef

or do even more ridiculous things:

```
\baselineskip=0pt\relax
\lineskiplimit=-100pt\relax
\halign{#&#\cr a&b\cr c&d\cr&\cr e&f\cr}
```

ef

This entails that we need to take \baselineskip and \lineskiplimit into account, using them to compute min-height (for normal \baselineskip) or height values (in case of sufficiently negative \lineskiplimit values) for the cell's HTML node.

9 Limitations

This brings us to the first insurmountable difference between TeX and CSS: *lines*. A line of text in TeX

[9] Conversely, scaling brackets properly with stretchy="true" seems to not work in Firefox as yet.

consists of individual character boxes with individual heights, widths and depths, and the spacing between lines is governed by the three parameters `\baselineskip` (the "default" distance between two baselines), `\lineskiplimit` (the minimally allowed distance between the bottom of a line and the top of the subsequent one), and `\lineskip` (the minimal skip to insert between two lines, if their distance is below `\lineskiplimit`). In particular, the height of a horizontal box containing e.g. a single character is entirely determined by the height of that particular character.

In contrast, a line of text in HTML/CSS has a *fixed* height of the current `line-height` value regardless of the occurring characters — and every single character counts as a "line": for every character, a *leading* space is inserted on top of it to make the containing box adhere to the `line-height`. This makes box manipulation on the level of individual characters currently (almost) impossible.[10]

One striking example for this is the `\LaTeX` macro, where the `A` is enclosed in a `\vbox`. RusTeX replaces its expansion by a simple `\raise\hbox` to achieve the (almost) same effect.

Situations where the layout critically depends on very precise positioning and sizing of boxes remains tricky. This is the case, for example, with the `tikzcd` package, where the nodes are laid out as tables, with `pgf` arrows between the individual cells.

On another front, various macros make use of LaTeX floats in non-trivial ways, such as `\marginpar` and the `{wrapfig}` environment, making special treatment for them (as of yet) unavoidable.

Finally, the `xy` package is a clear example of where, due to its usage of custom fonts, there is currently no feasible way to achieve support in terms of TeX primitives alone; anecdotally, I have been told that a `pgf` driver for `xy` is in the works, which, if completed, would likely immediately work for RusTeX as well.

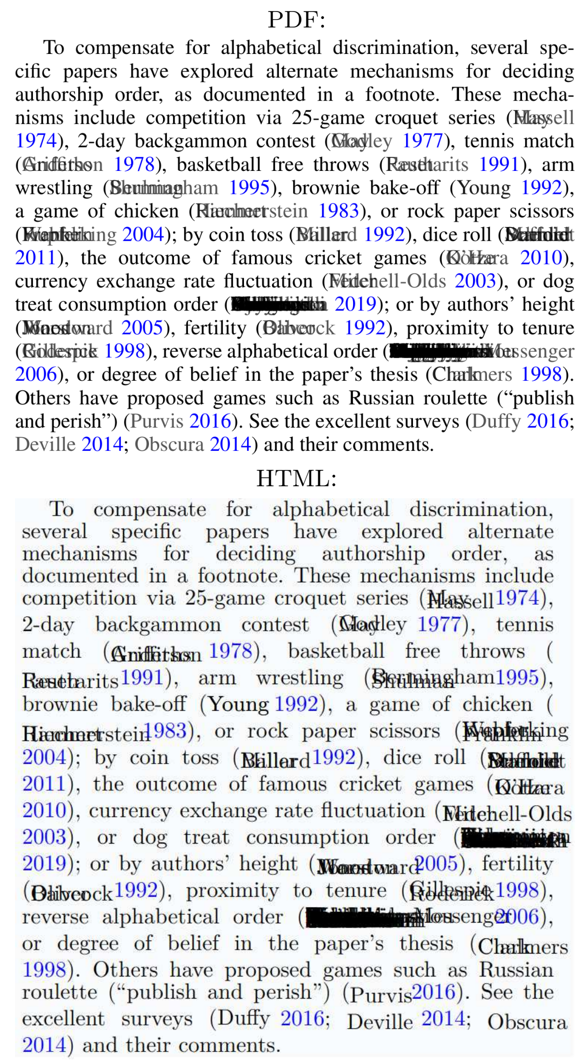
10 Conclusion

Despite the limitations mentioned above, the schema presented here works surprisingly well in a variety of cases. For example, list environments (`{itemize}`, `{enumerate}`, etc.), `{lstlisting}`, `{algorithmic}`, `tcolorbox`, figures, various environments for definitions, theorems and examples, `bibtex` and `biblatex`, and many other macros, environments and packages, often with intricate options and configurations, work out of the box without special treatment and with the expected presentation in the HTML.

Indeed, it is certainly surprising how much can be achieved without providing dedicated implementations for non-primitive macros, to the point where I am nowadays more surprised if the schema *fails* than when it *succeeds*.

To mention one particular highlight: A tongue-in-cheek paper was published in May 2023 on `arxiv.org` that argued for solving the order-of-authors problem in scientific publishing by *overlaying all the author names on top of each other*, including instructions how to achieve that in both TeX and HTML [1].

Running RusTeX over the LaTeX sources for the paper produced the right layout directly (Figure 1).


PDF:

To compensate for alphabetical discrimination, several specific papers have explored alternate mechanisms for deciding authorship order, as documented in a footnote. These mechanisms include competition via 25-game croquet series ([illegible] 1974), 2-day backgammon contest ([illegible] 1977), tennis match ([illegible] 1978), basketball free throws ([illegible] 1991), arm wrestling ([illegible] 1995), brownie bake-off (Young 1992), a game of chicken ([illegible] 1983), or rock paper scissors ([illegible] 2004); by coin toss ([illegible] 1992), dice roll ([illegible] 2011), the outcome of famous cricket games ([illegible] 2010), currency exchange rate fluctuation ([illegible] 2003), or dog treat consumption order ([illegible] 2019); or by authors' height ([illegible] 2005), fertility ([illegible] 1992), proximity to tenure ([illegible] 1998), reverse alphabetical order ([illegible] 2006), or degree of belief in the paper's thesis ([illegible] 1998). Others have proposed games such as Russian roulette ("publish and perish") (Purvis 2016). See the excellent surveys (Duffy 2016; Deville 2014; Obscura 2014) and their comments.

HTML:

To compensate for alphabetical discrimination, several specific papers have explored alternate mechanisms for deciding authorship order, as documented in a footnote. These mechanisms include competition via 25-game croquet series ([illegible] 1974), 2-day backgammon contest ([illegible] 1977), tennis match ([illegible] 1978), basketball free throws ([illegible] 1991), arm wrestling ([illegible] 1995), brownie bake-off (Young 1992), a game of chicken ([illegible] 1983), or rock paper scissors ([illegible] 2004); by coin toss ([illegible] 1992), dice roll ([illegible] 2011), the outcome of famous cricket games ([illegible] 2010), currency exchange rate fluctuation ([illegible] 2003), or dog treat consumption order ([illegible] 2019); or by authors' height ([illegible] 2005), fertility ([illegible] 1992), proximity to tenure ([illegible] 1998), reverse alphabetical order ([illegible] 2006), or degree of belief in the paper's thesis ([illegible] 1998). Others have proposed games such as Russian roulette ("publish and perish") (Purvis 2016). See the excellent surveys (Duffy 2016; Deville 2014; Obscura 2014) and their comments.

Figure 1: Screenshots from [1] in PDF and RusTeX generated HTML

[10] A proposal to the W3C CSS working group regarding leading space, which would presumably help here, has been open since 2018: `github.com/w3c/csswg-drafts/issues/3240`.

The most important aspects for generating adequate (and often great) HTML seem to be the "proper" treatment of `\hbox`/`\vbox`, `\hrule`/`\vrule` and skips/kerns, which RusTeX implements as described here. Their treatment should be relatively easily adaptable to, and usable by, other HTML converters, where "PDF-like" HTML output is desirable.

The most dire *limitations* are often related to intrinsic limitations of CSS — presumably, any extension of CSS that allows for more fine-grained control, especially on the character level, would allow for even better translations from TeX.

One more important limitation is the lack of accessibility features in the generated HTML, as a result of RusTeX operating on TeX primitives. With the adoption of the `tagpdf` package,[11] this can likely be easily remedied by providing primitive support for the annotations generated by it and translating them to corresponding HTML attributes.

Future work. Naturally, some of the techniques described here have been slightly simplified and are augmented in practice via various heuristics that are still subject to experimentation and improvements. Other discrepancies or problems are usually addressed (if possible) as we become aware of them (which still happens regularly).

It should be noted that RusTeX itself grew out of a hobby project, in the course of which I had to learn both Rust and TeX. As a result, the code base is, in hindsight, not well-structured and incompatible with various extensions that would be desirable; one example being support for XeTeX, which operates on Unicode rather than bytes for its basic character tokens. For that reason, I am currently working on refactoring the project into a modular library generic in as many aspects as possible, to allow for easily exchanging and adapting components of both the core engine and the output.[12] In the course of that, I am also experimenting with retaining the precise fonts used in a document, replacing them with their publicly available Unicode fonts where those exist.

Acknowledgements

The presented research is part of the VoLL-KI project, supported by the Bundesministerium für Bildung und Forschung (BMBF) under grant 16DHBKI089.

[11] `github.com/u-fischer/tagpdf`
[12] Published here: `crates.io/crates/tex_engine`

References

[1] E.D. Demaine, M.L. Demaine. Every author as first author, 2023. `arxiv.org/abs/2304.01393`.

[2] D. Fuchs. TeX font metric files. *TUGboat* 2(1):53–61, 1981. `tug.org/TUGboat/tb02-1/tb02fuchstfm.pdf`

[3] D.E. Knuth. *The TeXbook*. Addison-Wesley, 1984.

[4] M. Kohlhase, D. Müller. The sTeX3 package collection. `github.com/slatex/sTeX/blob/main/doc/stex-doc.pdf`

[5] J. MacFarlane. Pandoc — a universal document converter. `pandoc.org`, 2023.

[6] MathJax: Beautiful math in all browsers. `mathjax.org`

[7] B. Miller. LaTeXML: A LaTeX to XML converter. `dlmf.nist.gov/LaTeXML`

[8] D. Müller. *Mathematical Knowledge Management Across Formal Libraries*. Ph.D. thesis, Informatics, FAU Erlangen-Nürnberg, Dec. 2019. `opus4.kobv.de/opus4-fau/files/12359/thesis.pdf`

[9] D. Müller, et al. slatex/RusTeX. `github.com/sLaTeX/RusTeX`

[10] D. Müller, M. Kohlhase. sTeX3 — a LaTeX-based ecosystem for semantic/active mathematical documents. *TUGboat* 43(2):197–201, 2022. `tug.org/TUGboat/tb43-2/tb134mueller-stex3.pdf`. `kwarc.info/people/dmueller/pubs/tug22.pdf`

[11] L. Padovani. MathML formatting with TeX rules, TeX fonts, and TeX quality. *TUGboat* 24(1):53–61, 2003. `tug.org/tugboat/tb24-1/padovani.pdf`

[12] TeX4ht. `tug.org/tex4ht/`

⋄ Dennis Müller
Friedrich-Alexander University, Erlangen-Nürnberg, DE
`dennis.mueller (at) fau.de`
ORCID 0000-0002-4482-4912

Cheats (or not): When `\prevdepth = -1000pt`

Hans Hagen

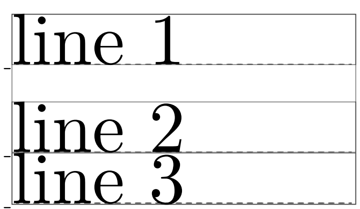
There are numerous quantities that a user can set in TeX, for example `\parskip` and `\parindent`. These are internal registers, and being registers they can be manipulated with for instance `\advance`. The more curious `\prevdepth` and `\prevgraf` are not registers but are properties of the current list. They are set by the engine but users can also set them in order to control or even fool the machinery. Here we focus on `\prevdepth`.

The depth of a box is normally positive but rules can have a negative depth in order to get a rule above the baseline. When TeX was written the assumption was that a negative depth of more than 1000 points made no sense at all. The last depth on a vertical list is registered in the `\prevdepth` variable. This is essentially a reference into the current list.

In order to illustrate some interesting side effects of setting this `\prevdepth`, and especially when we set it to −1000pt, this special sentinel value can be changed in LuaMetaTeX. However, as dealing with the property is somewhat special in the engine, you should not set it unless you know that the macro package is aware of it.

```
line 1\par line 2\par\nointerlineskip line 3\par
```

Assuming that we haven't set any inter-paragraph spacing this gives:



Here `\nointerlineskip` is (normally) defined as:

```
\def\nointerlineskip{\prevdepth-1000pt }
```

TeX also internally sets `\prevdepth` to −1000pt at the beginning of a vertical list, or just after a rule, to automatically suppress the next interline glue at those places.

In LuaMetaTeX we made all these hard-coded numbers configurable and this −1000pt is one of them. One reason for this is that it makes it easier to explain some of the side effects. When I tried to explain this to a curious user it was no fun to show pages spanning thousands of points. The variable that we can set is named `\ignoredepthcriterion`; in LMTX, that variable is used when we need to check for the magical value of `\prevdepth`.

We are now ready to give a more extensive example (`\ruledhbox` is a ConTeXt command):

```
\ruledhbox \bgroup
    \PrevTest{-10.0pt}\quad
    \PrevTest{-20.0pt}\quad
    \PrevTest{-49.9pt}\quad
    \PrevTest{-50.0pt}\quad
    \PrevTest{-50.1pt}\quad
    \PrevTest{-60.0pt}\quad
    \PrevTest{-80.0pt}%
\egroup
```

The `\PrevTest` helper, to construct a box of a given depth (negative in our cases here), is defined as (first example):

```
\def\PrevTest#1%
  {\setbox0\ruledhbox{\strut$\tf#1$}%
   \dp0=#1
   \vbox\bgroup\hsize4em
     FIRST\par
     \unhbox0\par
     LAST\par
   \egroup}
```

or (second example)

```
\def\PrevTest#1%
  {\setbox0\ruledhbox{\strut$\tf#1$}%
   \dp0=#1
   \vbox\bgroup
     \ruledhbox{FIRST}\par
     \box0\par
     \ruledhbox{LAST}\par
   \egroup}
```

In this example we set `\ignoredepthcriterion` to −50.0pt instead of the normal −1000pt. The result is shown in figures 1 and 2. The first case is what we normally have in text; we haven't set `prevdepth` explicitly between lines, so TeX just looks at the depth of the lines. In the second case, where we typeset boxes instead of their contents, the depth is ignored when it is less than the criterion value which is why, when we set the depth of the box to a negative value, we get somewhat interesting skips.

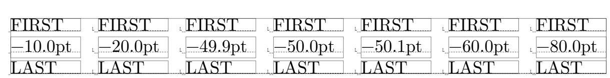


Figure 1: Showing explicitly-set depths of lines.

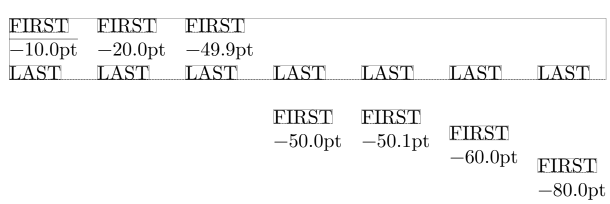


Figure 2: Depths above and below the magic `\prevdepth` value.

doi.org/10.47397/tb/44-2/tb137hagen-prevdepth

I'm sure one can use this effect in ways other than intended, but I doubt is any user is interested in doing so. Still, the fact that we can lower the criterion makes for nice experiments. For the record, in figure 3 you see what we get with positive values:

```
\ruledhbox \bgroup
    \PrevTest{10.0pt}\quad
    \PrevTest{20.0pt}\quad
    \PrevTest{49.9pt}\quad
    \PrevTest{50.0pt}\quad
    \PrevTest{50.1pt}\quad
    \PrevTest{60.0pt}\quad
    \PrevTest{80.0pt}%
\egroup
```

Watch the interline skip kick in when we make the depth larger than `\ignoredepthcriterion`, set here to (positive) 50pt. The (extremely) small additional space at 50.1pt is generated from `\lineskip`, while the full `\baselineskip` shows up at 60pt.

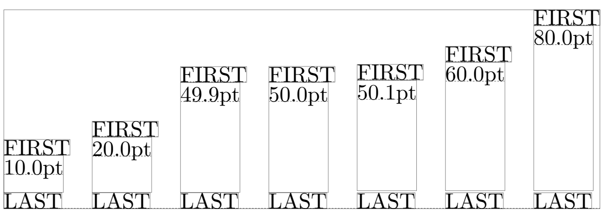


Figure 3: Positive depths.

So why don't we run into these side effects in regular documents? Simply because no one uses these excessive depths in box production, and also because hardly any user will work with such large paper sizes so for most users 1000pt (let alone −1000pt) is not something that they will naturally encounter.

This is true for many more mechanisms: sane usage is expected, so extreme cases can be ignored deep down in the engine. For instance, this is also why (due to old-times' performance reasons) wrap around of integers (and dimensions are just integers) is not that harmful and sometimes even useful, for instance when collecting content in boxes, where only when a user manipulates a related dimension some checking for overflow happens. Let's call them handy side effects.

Appendix: Implementation

For the record, here is the relevant snippet from original TEX:

```
@d ignore_depth==-65536000
       {|prev_depth| value that is ignored}
...
if prev_depth > ignore_depth then begin
  d := width(baseline_skip) - prev_depth
       - height(b);
  if d < line_skip_limit then
    p := new_param_glue(line_skip_code)
  else begin
    p := new_skip_param(baseline_skip_code);
    width(temp_ptr) := d;
  end;
  link(tail) := p;
  tail := p;
end;
link(tail) := b;
tail := b;
prev_depth := depth(b);
```

What we did was to make the `ignore_depth` constant into something like `int_par(ignore_depth_criterion_code)`.

If you look at the LuaMetaTEX code you will find something similar to the above, but there we split into functions. We also interface to a bit more granular paragraph (property) management, as well as adding callbacks.

◇ Hans Hagen
Pragma ADE

A roadmap for universal syllabic segmentation

Ondřej Sojka, Petr Sojka, Jakub Máca

Abstract

Space- and time-effective segmentation (word hyphenation) of natural languages remains at the core of every document rendering system, be it TEX, web browser, or mobile operating system. In most languages, segmentation mimicking syllabic pronunciation is a pragmatic preference today.

As language switching is often not marked in rendered texts, the typesetting engine needs *universal syllabic segmentation*. In this article, we show the feasibility of this idea by offering a prototype solution to two main problems:

A) Using Patgen to generate patterns for several languages at once; and

B) no wide character support in tools like Patgen or TEX hyphenation, e.g. internal Unicode support is missing.

For A), we have applied it to generating universal syllabic patterns from wordlists of nine syllabic, as opposed to etymology-based, languages (namely, Czech, Slovak, Georgian, Greek, Polish, Russian, Turkish, Turkmen, and Ukrainian). For B), we have created a version of Patgen that uses the Judy array data structure and compared its effectiveness with the trie implementation.

With the data from these nine languages, we show that:

A) developing universal, up-to-date, high-coverage, and highly generalized universal syllabic segmentation patterns is possible, with high impact on virtually all typesetting engines, including web page renderers; and

B) bringing wide character support into the hyphenation part of the TEX suite of programs is possible by using Judy arrays.

1 Motivation

Justified alignment achieved with a quality hyphenation algorithm is both optically pleasing and saves time to read, in addition to saving trees. Only *quality hyphenation* allows interword spaces to be as uniform as possible, close to Gutenberg's ideal of spaces of fixed width. A high coverage, space- and time-effective hyphenation (segmentation) algorithm of all natural languages is badly needed[1] as it remains at the core of every document rendering system, be it TEX, web browsers supporting HTML with CSS3, or an operating system providing text rendering for mobile applications.

In most languages, segmentation mimicking syllabic pronunciation is pragmatically preferred today. As language switching is often not marked in texts, and cannot be safely guessed from the words themselves, language-agnostic orthographic syllabification, is needed. We call this task *universal syllabic segmentation*, or in short, the syllabification problem.

The syllabification problem has been tackled by several finite state [2] or, more recently, machine learning techniques [1, 11, 14, 22]. Bartlett et al. [1] uses structured support vector machines (SVM) to solve syllabification as a tagging problem. Krantz et al. [6] leverage modern neural network techniques with long short-term memory (LSTM) cells, a convolutional component, and a conditional random field (CRF) output layer, and demonstrated cross-linguistic generalizability, syllabifying English, Dutch, Italian, French, Manipuri, and Basque datasets together.

From an orthographic viewpoint (hyphenation), universal language solutions today should reflect the Unicode standard [21]. Internal support for full Unicode, a must in today's operating systems and applications, *is missing* in the TEX family of programs, e.g. in Patgen and TEX itself. The internal processing is thus limited by the internal one-byte representation of language characters and is hardwired into the optimized code of these programs. Therefore, processing languages with huge character repertoires (Chinese, Japanese, Korean) and sets of languages whose character representations need *wide character* support is close to impossible. Special "hacks" are needed for character and font encodings both on the input side (package `inputenc`) and output side (packages `fontenc` or `fontspec`) are not backed by internal wide character support.

Since both TEX and Patgen have hardwired 8-bit character representations, to develop practically useable universal syllabic hyphenation, one needs to overcome these constraints.

In this paper we a) constructively show the feasibility of preparation of universal syllabic patterns, b) demonstrate a version of Patgen with wide character support, and c) discuss further steps to do in the TEX program suite to make language hyphenation Unicode-compliant.

The paper is structured as follows. In Section 2 we define the terminology and describe the language data we have used in our experiments. Section 3 reminds the reader about the principles of the hyphenation algorithm in TEX and of Patgen-based pattern generation and pattern representation possibilities.

[1] `bugzilla.mozilla.org/show_bug.cgi?id=672320`

doi.org/10.47397/tb/44-2/tb137sojka-syllabic

Table 1: Language resources and patterns used in pattern development experiments. All data was converted to UTF-8 and contains lowercase alphabetic characters only. Alphabet size (# chars) counts characters appearing in the language wordlist collected. Languages were chosen for diversity of size of patterns and syllables.

Language	# words	# chars	# patterns	# syllables	pattern source, alphabet
Czech+Slovak (cz+sk)	606,499	47	8,231	2,288,413	[19] correct optimized parameters, Latin
Georgian (ka)	50,644	33	2,110	224,799	[13] tex-hyphen repo, Georgian
Greek (el-monoton)	10,432	48	1,208	37,736	[13] tex-hyphen repo, Greek
Panjabi (pa)	892	52	60	2,579	[13] tex-hyphen repo, Gurmukhi
Polish (pl)	20,490	34	4,053	65,510	[13] tex-hyphen repo, Latin
Russian (ru)	19,698	33	4,808	75,532	[13] tex-hyphen repo, Russian
Tamil (ta)	46,526	48	71	209,380	[13] tex-hyphen repo, Tamil
Telugu (te)	28,849	66	72	125,508	[13] tex-hyphen repo, Telugu
Thai (th)	757	64	4,342	1,185	[13] tex-hyphen repo, Thai
Turkish (tr)	24,634	32	597	103,989	[13] tex-hyphen repo, Latin
Turkmen (tk)	9,262	30	2,371	33,080	[13] tex-hyphen repo, Latin
Ukrainian (ua)	17,007	33	1,990	65,099	[13] tex-hyphen repo, Cyrillic

Section 4 evaluates the experiments with universal pattern generation. In Section 5 we elaborate on possible routes towards wide character support in the typesetting engines and Patgen. As usual, we sum up and conclude in the final Section 6.

"The concept of the syllable is cross-linguistic, though formal definitions are rarely agreed upon, even within a language. In response, data-driven syllabification methods have been developed to learn from syllabified examples. ... Syllabification can be considered a sequence labeling task where each label delineates the existence or absence of a syllable boundary." [6]

2 Syllabification

Human beings convey meaning by pronouncing words as sequences of phonemes. Phonology studies the structure of phonemes we are able to pronounce as syllables [10]. Etymologically, a syllable is an Anglo-Norman variation of Old French sillabe, from Latin syllaba, from Greek συλλαβή (syllabē), "that which is held together; a syllable, several sounds or letters taken together" to make a single sound. [3]

When we delineate boundaries in the orthographic representation of words, we speak about *hyphenation* of words as sequences of *characters*.

2.1 Hyphenation as syllabification

There are subtle differences between syllabification and hyphenation, though. Let us take the Czech word *sestra*. The Czech language authorities [23] allow hyphenations as *se-s-t-ra*, while agreeing that there are only two syllables based on **C**onsonant and **V**owel sequencing: either *se-stra* (CV-CCCV), or *ses-tra* (CVC-CCV), or *sest-ra* (CVCC-CV). As with hyphenation, defining segments for syllabification is full of exceptions. The Czech sentence *Strč prst skrz krk* or word *scvrnkls* (CCCCCCCC) contain consonants-only syllables.

There are also rare cases where word segmentation should differ in different contexts. It may be necessary within one language (different hyphenation *re-cord* and *rec-ord* depending on its part of speech), or between different languages. When developing universal syllabic patterns, these theoretically possible segmentations should not be allowed in the input hyphenated wordlist used for training. But this should not matter, as e.g. Liang's `hyphen.tex` patterns do not cover more than 10% of positions [8] and few complain about this coverage.

2.2 Data preparation

To show the feasibility of universal pattern generation, we have collected wordlists for a dozen languages, as shown in Table 1. The chosen languages a) have a wide diversity in alphabets and syllables and b) have existing hyphenation patterns as an approximation for syllable segments. The wordlists were collected from public sources or provided for our research as stratified dictionaries from TenTen corpora [4] by Lexical Computing. We used wordlists sorted by frequency and cut at below 5% of word occurrences, to eliminate typos appearing in documents. Each tenth word was taken into a wordlist — a stratified sampling technique inspired by Knuth [5] that was already used successfully in pattern generation [20]. Wordlists were hyphenated by legacy patterns, mostly taken from [13].

Table 2: Language alphabet overlaps. Cells contain the number of lowercase letters that overlap between languages. In total, 13 languages contain in total 412 different lowercase letters, more than Patgen is capable of digesting.

Language	cz+sk	ka	el	pa	pl	ru	ta	te	th	tr	tk	ua
Czech+Slovak (cz+sk)	47	0	0	0	26	0	0	0	0	25	28	0
Georgian (ka)	0	33	0	0	0	0	0	0	0	0	0	0
Greek (el-monoton)	0	0	48	0	0	0	0	0	0	0	0	0
Panjabi (pa)	0	0	0	52	0	0	0	0	0	0	0	0
Polish (pl)	26	0	0	0	34	0	0	0	0	23	22	0
Russian (ru)	0	0	0	0	0	33	0	0	0	0	0	29
Tamil (ta)	0	0	0	0	0	0	48	0	0	0	0	0
Telugu (te)	0	0	0	0	0	0	0	66	0	0	0	0
Thai (th)	0	0	0	0	0	0	0	0	64	0	0	0
Turkish (tr)	25	0	0	0	23	0	0	0	0	32	25	0
Turkmen (tk)	28	0	0	0	22	0	0	0	0	25	30	0
Ukrainian (ua)	0	0	0	0	0	29	0	0	0	0	0	33

Alphabet analysis and statistics are shown in Table 2. The total number of characters appearing in all languages exceeds 245, the maximum number of characters that current Patgen can support. This is why wide-character representation (Unicode UCS-2) support in Patgen (and then in the hyphenator library in a typesetting engine) would be needed to extend our generation to more languages.

3 Pattern development

The idea of squeezing the hyphenated wordlist into the set of patterns was originated in the dissertation of Frank Liang [8], supervised by Donald Knuth. For the automated generation of patterns from a wordlist, Liang wrote the Patgen program. Patgen was one of the very first programs that harnessed the power of data with supervised machine learning. Programmed originally to support English and ASCII, it was later extended to be usable for 8-bit characters and for wordlists that contain at most 245 characters [9]. It is capable of efficient lossy or lossless *compression* of hyphenated dictionaries, with several orders of magnitude compression ratio. Generated patterns have minimal length, e.g., the shortest context possible, which results in their *generalization* properties.

In general, *exact lossless* pattern *minimization is non-polynomial* by reduction to the minimum set cover problem [16]. For Czech, *exact lossless* pattern generation *is feasible* [17], while reaching *100% coverage* and simultaneously *no errors*. Strict pattern minimality (size) is not an issue nowadays.

This idea and its realization is a programming pearl. Motivated by space and time constraints, instead of the classical solution of dictionary problem in the logarithmic time of dictionary size, the word hyphenation is computed from patterns in constant time, where the constant is given by *word* length.

Space needed for patterns in the *packed trie* data structure is typically in tens of kB, which is several orders of magnitude smaller than the wordlist size. With fine-tuned parameters of pattern generation in the so-called *levels*, one can prepare patterns with zero errors and almost full coverage of hyphenation points from the input dictionary.

For practical use, patterns are collected in the repository maintained by the TEX community [13]. It is no surprise that most if not all leading typesetting engines deploy this "competing pattern engineering technology" [15].

3.1 Patterns

The patterns "compete" with each other whether to split the word at a position, given varying characters in both side contexts; see Figure 1.

We have shown how effective and powerful the technique is, and that its power depends on the *parameters* of pattern generation [17]. The key is the proper setting of Patgen parameters for pattern generation. The idea of universal segmentation with Patgen has been proposed already in [18]. There, we demonstrated the techniques for the development of two languages together, Czech and Slovak, and developed a joint wordlist and patterns [19].

We wanted to extend the technique to other Slavic and syllabic languages. The bottleneck for adding new languages was Patgen and TEX's constraint of one-byte character support only for storing patterns in tries. We thought of using a modern data structure that would allow wide character trie

```
    h y p h e n a t i o n
p1           1n a
p1               1t i o n
p2            n2a t
p2                 2i o
p2        h e2n
p3  h y3p h
p4        h e n a4
p5        h e n5a t
    h0y3p0h0e2n5a4t2i0o0n
```

hy-phen-ation → 2 6
... → ...
... → ...
key → data

Solution to the dictionary problem:
For key part (the word) to store
the data part (its division)

Figure 1: Eight patterns "compete" how to hyphenate *hyphenation*. Winners are patterns `hy3ph` and `hen5at` generated at the highest covering level (odd numbers) generation. The level hierarchy allows for storing exceptions, exceptions to exceptions, exceptions to exceptions to exceptions, ..., with character contexts as parameters. [8]

representation. That was the task for a bachelor's thesis: use a Judy array [12].

3.2 Judy arrays

The Judy array, also known as simply Judy, is a data structure that implements a sparse dynamic array, allowing for versatile applications such as dynamically-sized arrays and associative arrays. Judy is internally implemented as a tree structure, where every internal node has 256 ancestor nodes. The most interesting thing about this structure is that it tries to be as memory-efficient as possible by effectively using available cache, avoiding unnecessary access to main memory. As a result, Judy is both fast and memory-efficient.

The feasibility of utilizing the Judy structure for storing hyphenation patterns is demonstrated in the thesis [12]. In Chapter 4, it is shown that Judy has the potential to be faster and more memory-efficient compared to the original trie when working with patterns. Further, Chapter 5 explores the potential integration of Judy into Patgen and the consequent impact on Patgen's generation process. The results from this chapter indicate that rewriting Patgen with Judy is possible but would require an almost complete overhaul of Patgen's code and algorithms. This redevelopment would yield a Patgen version capable of handling input of any kind, enabling the generation of patterns composed of arbitrary alphabets. However, it is important to note that the generation process would be approximately four times slower than the current implementation. This is due to the hiding of access to the inner nodes of stored tries in Judy. As this access is not needed in TEX for the hyphenation of individual words, using some variant of Judy in a TEX successor would make hyphenation faster.

3.3 Universal pattern generation

To pursue the idea of universal syllabic pattern generation, we have checked whether the legacy patterns hyphenate the same valid word in different languages differently. The result with a short discussion is in Table 3. The expectation that syllable-forming principles are universal, as phonology theory suggests, is confirmed. The errors we have found were due to the difference between hyphenation and syllabification caused by inconsistent markup rather than a principled difference in word morphology, e.g. a compound word segmented in one language, and given as a single word in the other.[2]

We removed all colliding words when joining wordlists into the wordlist universal pattern generation. As mentioned earlier, we collected words for nine languages (cz, sk, ka, el, pl, ru, tr, tk, ua).

We generated universal patterns with the same three sets of Patgen parameters (custom, correct optimized, and size optimized) as when generating Czechoslovak patterns. The results are shown in Tables 4 (custom), 5 (correct optimized) and 6 (size optimized). The results are comparable with generation for two languages and confirm the feasibility of universal pattern development.

We did not pursue 100% coverage at all costs because the source data is noisy, and we do not want the patterns to learn all the typos and inconsistencies. Also, the size of the new languages was rather small, compared to Czechoslovak.

4 Evaluation

We evaluated the quality of developed patterns by two metrics. *Coverage* of hyphenation points in the training wordlist tells how the patterns correctly

[2] Compound words can evolve in perception into single words even within one language. Examples are the evolution of *e-mail* into *email* or *roz-um* into syllabic *ro-zum* in Czech.

Table 3: Different word hyphenation overlaps. Cells contain the number of same words that are segmented differently between languages. Differences are caused typically by suboptimal coverage patterns used to hyphenate the wordlist (*vi-bram* vs. *vib-ram*, *up-gra-de* vs. *upg-ra-de*). We remove the differently hyphenated words when joining wordlists for the final syllabic generation.

Language	cz+sk	ka	el	pa	pl	ru	ta	te	th	tr	tk	ua
Czech+Slovak (cz+sk)	9	0	0	0	388	0	0	0	0	640	69	0
Georgian (ka)	0	0	0	0	0	0	0	0	0	0	0	0
Greek (el-monoton)	0	0	0	0	0	0	0	0	0	0	0	0
Panjabi (pa)	0	0	0	0	0	0	0	0	0	0	0	0
Polish (pl)	388	0	0	0	0	0	0	0	0	187	9	0
Russian (ru)	0	0	0	0	0	0	0	0	0	0	0	125
Tamil (ta)	0	0	0	0	0	0	0	0	0	0	0	0
Telugu (te)	0	0	0	0	0	0	0	0	0	0	0	0
Thai (th)	0	0	0	0	0	0	0	0	0	0	0	0
Turkish (tr)	640	0	0	0	187	0	0	0	0	0	80	0
Turkmen (tk)	69	0	0	0	9	0	0	0	0	80	0	0
Ukrainian (ua)	0	0	0	0	0	125	0	0	0	0	0	0

predicted hyphenation points used in training. *Generalization* means how the patterns behave on unseen data, on words not available in the data used during Patgen training. The methodology is the same as we used in the development of Czechoslovak patterns [19].

In Table 7, we compare the efficiency of different approaches to hyphenating 2 languages and 9 languages from one pattern set. We see that the performance of universal patterns is comparable in size and quality to double- or single-language ones — there is only a negligible difference. Table 8 shows that generalization qualities, given the small input size wordlists, are very good, and comparable to the fine-tuned Czechoslovak results. Investing in the purification and consistency of input wordlists (as we did for Czech and Slovak) would result in near-perfect syllabic patterns with almost 100% coverage and no errors.

5 Future work

A natural further step is to merge further languages where the syllabic principle is used for hyphenation. For that, one would need a version of Patgen we provisionally call UniPatgen. This version would support Unicode not only in I/O but also internally as a wide character (UCS-2) character encoded in the pattern representation in either a packed trie or Judy array. This would allow merging more languages *without* increasing the computational complexity of hyphenation, and only a sublinear increase of pattern size. We believe that coverage may differ from 100% only by words that should be hyphenated differently in different languages — our estimate is in small, single-digit percents, while, as mentioned above, the widely-used `hyphen.tex` patterns do not cover 10+%!

Another possible extension in pattern development is the support of a specific hyphenation penalty for compound word borders. This extension, discussed already 30 years ago [20], would generate patterns first for compound words, and only after fixing them continue with pattern generation for all other hyphenation points. The TeX engine would then set the hyphenation penalties depending on level ranges in patterns found for the hyphenated word. This extension is orthogonal with support for universal patterns but might require increasing the maximal number of levels allowed in patterns to two digits.

There are several open questions for the TeX development community:

1. Should the universal syllabic patterns ever be developed?
2. If so, should the needed *internal* wide character representations be added to the TeX suite of programs? That is, to TeX-based engines not yet supporting it[3] and Patgen or UniPatgen.
3. If not, should it be handled by external segmenters on TeX's input, based on Patgen's proposed successor, UniPatgen?

[3] `cs.overleaf.com/learn/latex/TeX_primitives_listed_by_TeX_engine`

Table 4: Statistics from the generation of universal patterns for cz+sk, ka, el, pl, ru, tr, tk, ua with *custom* parameters and `\lefthyphenmin=2`, `\righthyphenmin=2`. Generation took 33.23 seconds, 11,238 patterns, 77 kB.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	2,407	2,066,410	280,020	70,588	1 3	1 3 12
2	2,375	2,025,245	8,866	111,753	2 4	1 1 5
3	4,626	2,118,063	19,213	18,935	3 6	1 2 4
4	2,993	2,117,739	5,920	19,259	3 7	1 4 2

Table 5: Statistics from the generation of universal patterns for cz+sk, ka, el, pl, ru, tr, tk, ua with *correct optimized* parameters and `\lefthyphenmin=2`, `\righthyphenmin=2`. Generation took 35.43 seconds, 29,742 patterns, 219 kB.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	7,188	2,049,375	164,224	87,623	1 3	1 5 1
2	4,108	2,042,249	14,094	94,749	1 3	1 5 1
3	15,010	2,134,692	20,544	2,306	2 6	1 3 1
4	6,920	2,133,458	815	3,540	2 7	1 3 1

Table 6: Statistics from the generation of universal patterns for cz+sk, ka, el, pl, ru, tr, tk, ua with *size optimized* parameters and `\lefthyphenmin=2`, `\righthyphenmin=2`. Generation took 29.75 seconds, 14,321 patterns, 101 kB.

Level	Patterns	Good	Bad	Missed	Lengths	Params
1	1,201	2,092,928	598,321	44,070	1 3	1 2 20
2	2,695	1,736,372	5,274	400,626	2 4	2 1 8
3	4,835	2,102,803	20,094	34,195	3 5	1 4 7
4	6,508	2,099,607	210	37,391	4 7	3 2 1

4. If UniPatgen was developed, should it be added to the distribution, together with Unicode patterns included and supported in repositories like [13]?
5. Should UniPatgen, and LuaTEX, add a dependency on a Judy library, or should a more conservative solution be sought and implemented? With a conservative solution, which data structure to use for storing patterns? Should the memory be allocated dynamically, to overcome the abundant explosion of format size that stores the patterns, as output by iniTEX?
6. Should UniPatgen (and TEX engines) additionally and orthogonally support patterns and different hyphenation penalty for compound word borders, currently available in e.g. the German wordlist [7]?

We would appreciate qualified opinions on these decisions being sent to authors.

"All we are saying, give patterns a chance."
Our paraphrase of John Lennon's protest song refrain

6 Conclusion

Preparation of language-agnostic, i.e. universal, syllabic segmentation patterns could be done! We have demonstrated this possibility by generating patterns based on the wordlists of nine languages with current Patgen. They have superb generalization qualities, high coverage of hyphenation points (more than most legacy patterns), and virtually no errors. Their use could have a high impact on virtually all typesetting engines including web page renderers.

Supporting wide characters in Patgen is a critical requirement for adding more languages. We have shown that bringing wide character support into the hyphenation part of the TEX suite of programs is possible by using the Judy array. It will allow generating and deploying patterns for the whole Unicode character set. We have discussed a possible roadmap

Table 7: Comparison of the efficiency of different approaches to pattern generation of Czechoslovak and of universal patterns. Note that the size of universal patterns grows sublinearly with the number of languages. The generalization ability of universal patterns is only slightly worse than that of Czechoslovak ones. The experience from the development of Czechoslovak patterns shows that performance could be improved by consistent markup of wordlist data.

Wordlist	Parameters	Good	Bad	Missed	Size	Patterns
Czechoslovak	custom	99.87%	0.03%	0.13%	32 kB	5,907
Czechoslovak	correctopt	99.99%	0.00%	0.01%	45 kB	8,231
Czechoslovak	sizeopt	99.67%	0.00%	0.33%	40 kB	7,417
Universal	custom	99.10%	0.28%	0.90%	77 kB	11,238
Universal	correctopt	99.83%	0.04%	0.17%	219 kB	29,742
Universal	sizeopt	98.25%	0.01%	1.75%	101 kB	14,321

Table 8: Results of 10-fold cross-validation (learning on 90%, and testing on remaining 10%). Generalization properties (performance on words not seen during training) are compared with Czechoslovak patterns. By adding 7 languages, the generalization abilities of universal patterns are only slightly worse.

Wordlist	Parameters	Good	Bad	Missed
Czechoslovak	custom	99.64%	0.22%	0.14%
Czechoslovak	correctopt	99.81%	0.15%	0.04%
Czechoslovak	sizeopt	99.41%	0.18%	0.40%
Universal	custom	97.99%	1.06%	0.95%
Universal	correctopt	98.10%	1.28%	0.62%
Universal	sizeopt	97.50%	0.94%	1.56%

to make this a reality in typesetting engines including TeX successors.

Acknowledgments

We are indebted to Don Knuth for questioning the common properties of Czech and Slovak hyphenation during our presentation of [17] at TUG 2019, which has led us in this research direction. We also thank everyone on whose shoulders we build our work, e.g. for wordlists by Lexical Computing, and to all who commented on our work at TUG 2021 [19] and TUG 2023.

References

[1] S. Bartlett, G. Kondrak, C. Cherry. Automatic Syllabification with Structured SVMs for Letter-to-Phoneme Conversion. In *Proceedings of ACL-08: HLT*, pp. 568–576, Columbus, Ohio, June 2008. Assoc. for Computational Linguistics. `aclweb.org/anthology/P08-1065`

[2] Y. Haralambous. New hyphenation techniques in Ω_2. *TUGboat* 27(1):98–103, 2006. `tug.org/TUGboat/tb27-1/tb86haralambous-hyph.pdf`

[3] Online etymology dictionary. `"syllable"`. `www.etymonline.com/word/syllable`

[4] M. Jakubíček, A. Kilgarriff, et al. The TenTen Corpus Family. In *Proc. of the 7th International Corpus Linguistics Conference (CL)*, pp. 125–127, Lancaster, July 2013.

[5] D.E. Knuth. *3 : 16 Bible Texts Illuminated*. A-R Editions, Inc., 1991.

[6] J. Krantz, M. Dulin, P.D. Palma. Language-agnostic syllabification with neural sequence labeling. *CoRR* abs/1909.13362, 2019. `arxiv.org/abs/1909.13362`

[7] W. Lemberg. A database of German words with hyphenation information, 2023. `repo.or.cz/wortliste.git`

[8] F.M. Liang. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Dept. of Computer Science, Stanford University, Aug. 1983. tug.org/docs/liang/liang-thesis.pdf

[9] F.M. Liang, P. Breitenlohner. PATtern GENeration program for the TeX82 hyphenator. Electronic documentation of PATGEN program version 2.4 on CTAN. ctan.org/pkg/patgen, 1999.

[10] I. Maddieson. Syllable Structure. In *The World Atlas of Language Structures Online*, M.S. Dryer, M. Haspelmath, eds. Max Planck Institute for Evolutionary Anthropology, Leipzig, 2013. wals.info/chapter/12

[11] Y. Marchand, C.R. Adsett, R.I. Damper. Automatic Syllabification in English: A Comparison of Different Algorithms. *Language and Speech* 52(1):1–27, 2009. doi.org/10.1177/0023830908099881

[12] J. Máca. Judy, May 2023. Bachelor Thesis supervised by Petr Sojka and defended at Masaryk University, Faculty of Informatics. is.muni.cz/th/kru3j

[13] A. Rosendahl, M. Miklavec. TeX hyphenation patterns, 2023. Accessed 2023-07-05. http://hyphenation.org/tex

[14] Y. Shao, C. Hardmeier, J. Nivre. Universal Word Segmentation: Implementation and Interpretation. *Transactions of the Association for Computational Linguistics* 6:421–435, 2018. doi.org/10.1162/tacl_a_00033

[15] P. Sojka. Competing Patterns for Language Engineering. In *Proceedings of the Third International Workshop on Text, Speech and Dialogue—TSD 2000*, P. Sojka, I. Kopeček, K. Pala, eds., LNAI 1902, pp. 157–162, Brno, Czech Republic, Sept. 2000. Springer-Verlag. doi.org/10.1007/3-540-45323-7_27

[16] P. Sojka. *Competing Patterns in Language Engineering and Computer Typesetting*. Ph.D. thesis, Masaryk University, Brno, Jan. 2005. researchgate.net/publication/265246931_Competing_Patterns_in_Language_Engineering_and_Computer_Typesetting/

[17] P. Sojka, O. Sojka. The Unreasonable Effectiveness of Pattern Generation. *TUGboat* 40(2):187–193, 2019. tug.org/TUGboat/tb40-2/tb125sojka-patgen.pdf

[18] P. Sojka, O. Sojka. Towards Universal Hyphenation Patterns. In *Proceedings of Recent Advances in Slavonic Natural Language Processing—RASLAN 2019*, A. Horák, P. Rychlý, A. Rambousek, eds., pp. 63–68, Karlova Studánka, Czech Republic, 2019. Tribun EU. is.muni.cz/publication/1585259/?lang=en. nlp.fi.muni.cz/raslan/2019/paper13-sojka.pdf

[19] P. Sojka, O. Sojka. New Czechoslovak Hyphenation Patterns, Word Lists, and Workflow. *TUGboat* 42(2), 2021. doi.org/10.47397/tb/42-2/tb131sojka-czech

[20] P. Sojka, P. Ševeček. Hyphenation in TeX — Quo Vadis? *TUGboat* 16(3):280–289, 1995. tug.org/TUGboat/tb16-3/tb48soj1.pdf

[21] The Unicode Consortium. *The Unicode Standard: Worldwide Character Encoding. Version 15.0.* Unicode, Inc., Mountain View, CA, USA, 2022. unicode.org/versions/Unicode15.0.0

[22] N. Trogkanis, C. Elkan. Conditional Random Fields for Word Hyphenation. In *Proceedings of the 48th Annual Meeting of the ACL*, pp. 366–374, Uppsala, Sweden, July 2010. ACL. aclweb.org/anthology/P10-1038

[23] Internetová jazyková příručka (Internet Language Reference Book), 2023. prirucka.ujc.cas.cz/?id=135

◇ Ondřej Sojka,
Petr Sojka,
Jakub Máca
Faculty of Informatics, Masaryk University, Brno, Czech Republic
454904 (at) mail dot muni dot cz, sojka (at) fi dot muni dot cz, 514024 (at) mail dot muni dot cz
ORCID 0000-0003-2048-9977, 0000-0002-5768-4007, 0009-0008-1583-3183

METAFONT/METAPOST and a complex Indic script: Malayalam

C. V. Radhakrishnan, K. V. Rajeesh,
K. H. Hussain

Abstract

Malayalam is an Indic script with numerous shape-shifting characters. We explore a reusable component-based design for Malayalam fonts, and develop them using METAFONT/METAPOST [6, 1] to assemble the characters. We discuss the paradigm shift from GUI design tools to 'code-based' design of shapes and glyphs, even by non-coders, and the advantages and challenges of using METAFONT/METAPOST to develop an OpenType font for a complex script. Finally, the progress made by our small team is shared.

1 Indic scripts and Malayalam

The Union of India has 23 languages [11], each one being the official language of one or more states. For convenience of governance, the Union was divided into states comprising areas with people speaking the same language. Thus, Kerala is the state of people speaking Malayalam, the adjacent state of Tamil Nadu is that of Tamil-speaking people, Karnataka of Kannada-speaking people, and so on. Any of these 23 languages can be used for official communication, including deliberations in the Indian Parliament. The Indian currency note bears the denomination in all languages (see Figure 1; fewer entries are due to the fact that some of the languages share the same script, apart from Hindi and English, which are already on the face of the note).

The Brahmic scripts [10], the family of languages to which Malayalam belongs, have a few common properties among most of the members. Each consonant has an inherent vowel, which is usually a short '`a`', and other vowels are written by con-

Figure 1: The picture of the 200 rupees currency note of India; the denomination is printed in 15 scripts. (courtesy: Reserve Bank of India)

Figure 2: A few Malayalam characters to show the generally rounded shape.

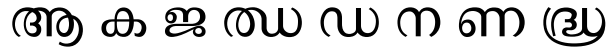
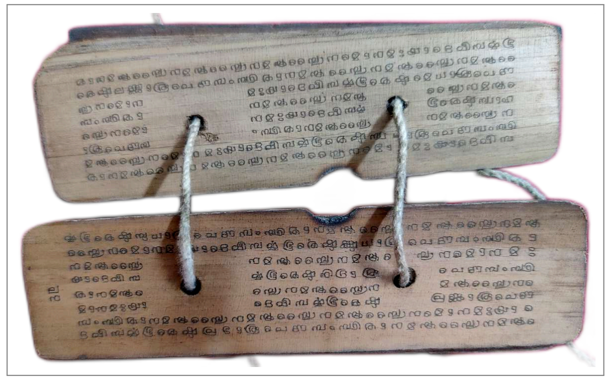


Figure 3: Manuscript leaves of Malayalam text. (courtesy: Wikipedia)



joining with the character. Each vowel has an independent form when not attached to a consonant, and a dependent form, attached to a consonant, at times to both on the left and right sides of the consonant. Up to four consonants can be combined in ligatures. Special marks are added to denote the combination of '`r`' with another consonant. Nasalization and aspiration of a consonant's dependent vowel are also denoted by separate signs. The alphabetical order is: vowels, velar consonants, palatal consonants, retroflex consonants, dental consonants, bilabial consonants, approximants, sibilants, and other consonants. Each consonant grouping has four stops (with all four possible values of voicing and aspiration, see Table 3) ending with a nasal consonant [10].

The above properties of the Brahmic family can be found in the Malayalam script. For instance, the first syllable in the word *poppler*, written in Malayalam, needs a vowel representation on both sides of the consonant *p*, which will look like 'പൊ' where the middle character (പ) represents *p* and those on the sides represent the vowel *o*. Thus, a Malayalam font table can be enormous in size, with over 900 glyphs that constitute basic vowels and consonants for forming 57 characters, while the rest constitute the ligatures, vertical and horizontal conjuncts derived from the basic character set. *RIT Rachana* [3], a popular font in Malayalam, has over 920 glyphs derived from a base font table comprising 117 characters in the Unicode font table [9]. The script occupies the code points between `00D0` and `0D7F` in the Unicode table.

doi.org/10.47397/tb/44-2/tb137radhakrishnan-malayalam

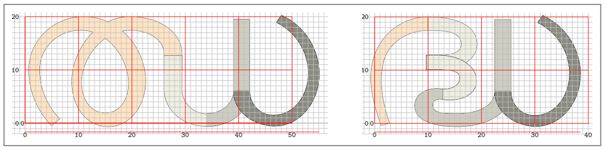


Figure 4: Two glyphs showing shape components used with different colors.

2 Rationale

Upon closer examination of the characters in the Malayalam script (refer to Figure 2), one will notice that the majority of these characters are composed of arcs, semicircles, and circles of varying sizes. These elements harmoniously come together to form the distinctive shape of each character. The origins of the rounded and cursive design of these letters can often be attributed to the writing materials that were used in the past.

It is believed that the prevalence of round and cursive shapes in the letters of Indic languages can be traced back to the practicality of the writing instruments employed during their origin. The traditional writing instrument was a long, sharp metallic stylus used for inscribing text on dried and smoked palm leaves. Angular shapes would have been unsuitable as they could potentially tear the delicate leaves as they were being written upon. Hence, it seems plausible that this practical consideration influenced the widespread adoption of round and cursive letter forms (see Figure 3 showcasing a manuscript as an example).

The inherent rounded and cursive nature of the letters naturally led to the conclusion to create a set of predefined components with specific shapes. These components could then be reused effectively to construct complete characters. As one can easily deduce, the concept of reusability not only saves considerable time and effort but also ensures a consistent and uniform quality in terms of curves, cut angles, and similar attributes. Nonetheless, it is essential to acknowledge that this approach does have its limitations, some of which will be discussed in the following sections pertaining to reusable components and glyphs (Sections 3.3 and 3.4).

These limitations have significantly influenced design decisions, resulting in the development of methods to manipulate coordinates, adjust widths, and alter angles in the components as needed to fit the overall shape of the character of which they are part. In Figure 4, which showcases two glyphs, you can observe the utilization of shape components, distinguished by different colors to ease comprehension and analysis.

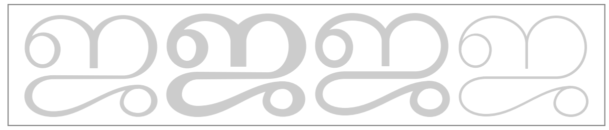


Figure 5: Four variants of a consonant character (pronounced *dja*) from the same METAPOST source code.

Another significant factor in our decision to utilize METAFONT/METAPOST was the potential for reusing the source code to generate variants of the font family. By employing separate configurations for each variant and requiring only minimal adjustments, we could easily create font variations of remarkable quality, thereby reducing both development time and effort. Four variants of a consonant character (pronounced *dja*), using the same METAPOST source code with different values for a few variables can be seen in Figure 5.

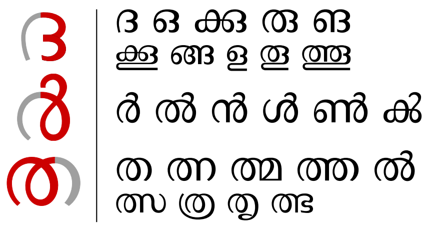
In the realm of character description languages, our choice of the code-based METAFONT/METAPOST can be attributed to our enduring association with the illustrious TeX [5] and its companions. We derive immense pleasure from employing TeX's sagacious and programmable markup language to fulfill our multifarious text processing requisites, eschewing the allure of graphical interface-driven applications. However, the merits of METAFONT/METAPOST extend far beyond mere preference.

The selection of METAFONT/METAPOST endows us with an array of supplementary advantages, including seamless cross-platform compatibility and the remarkable capability to produce vector outputs in the form of SVG and PostScript. Furthermore, the maintenance of our codebase becomes a simplified endeavor through the use of text-based source code, fostering clarity and facilitating future modifications. It is a confluence of these factors that harmoniously resonate with our intrinsic predilection for code-driven development, making the adoption of METAFONT/METAPOST an instinctive and judicious decision.

3 Research and design process

Once the design based on reusable components was finalized, the subsequent undertaking entailed identifying the most suitable curves and shapes to fulfill the objective. A comprehensive list of components was meticulously compiled, along with a corresponding catalog of potential characters and glyphs that could harness the potential of these components. To facilitate reader comprehension, a typeset list containing the components and the characters associated with them is provided in PDF format at

Figure 6: Reusable components and characters that can use them. The components are shown in red.



the link in [7]. In Figure 6, you will find three representative examples selected from this extensive list, providing a glimpse into the range of possibilities that await exploration.

Although the choice to delve into the realm of METAFONT/METAPOST came naturally to us, none of us possessed prior familiarity with the language or its intricacies. Consequently, our initial focus revolved around acquiring a firm grasp and cultivating a reasonable proficiency in the art of METAFONT/METAPOST. This endeavor demanded unwavering dedication and a significant investment of time, but it granted us the confidence necessary to embark on our creative journey.

As we delved deeper into the captivating world of METAFONT, we soon encountered a crucial turning point. After careful deliberation, we made the decision to transition to METAPOST. We recognized that METAPOST held a distinct advantage, enabling direct generation of vector outputs in the form of SVG and PostScript. These invaluable features seamlessly aligned with our ultimate objective of crafting fonts using software like FontForge or `fontmake`. Thus, our pursuit of perfection urged us to embrace the versatility and convenience offered by METAPOST, as it emerged as the perfect companion on our path towards mastering the art of font creation.

3.1 Initial attempts

Initially, our endeavors centered on creating the foundational characters in adherence to the Unicode table [9], employing fixed coordinates and distinct `penstroke` [6, p. 273] and `draw` [6, p. 271] functions for serif and sans-serif variants respectively. However, it wasn't long before we encountered the inherent limitations of this approach. Firstly, the fixed coordinate system presented a significant drawback, as it required users to manually input all the coordinates, contrasting with the more flexible algebraic expressions that would prompt METAFONT to calculate the coordinates through solving these expressions. The design of METAFONT itself encourages users to adopt the latter method, as it allows for greater parametrization and versatility.

The sans-serif variants in SVG format, generated through the draw function with uniform line thickness, were initially considered viable candidates. However, they exhibited potential flaws when it came to removing overlaps during the font creation process, whether through FontForge or applications like Inkscape. It became apparent that utilizing stroke commands such as `penstroke`, which enables drawing an envelope of specified thickness and angles around the central line, as dictated by `penpos` [6, p. 273] commands for each coordinate, would ensure seamless overlap removal. This realization prompted us to abandon fixed coordinates in favor of a more suitable approach.

3.2 Parametrized approach

Various factors that impact the shape, angle, and thickness of strokes have been successfully parameterized. These encompass a range of essential parameters, as well as supplementary parameters that rely on the values of the foundational ones. A comprehensive listing of these parameters, including both the fundamental and dependent ones, can be found in Tables 1 and 2.

The parameters `t` and `u` play a crucial role in finely adjusting dimensions, although they differ in their impact. While both parameters contribute to this adjustment process, it's worth noting that `t` has the unique characteristic of having no effect on sans-serif variants. In other words, its influence becomes apparent only when serif variants come into play.

In addition to the aforementioned parametrization, we have also undertaken another set of parameter adjustments concerning the widths and angles of strokes, tailored to accommodate various variants. Initially, our focus encompassed four primary variants: serif, sans-serif, serif thin, and sans-serif thin. However, it is important to note that we have the flexibility to incorporate additional variants in the future, should the need arise.

To facilitate this parametrization process, we defined a numeric variable, `width_angle`, with val-

Table 1: The essential/foundational parameters used.

Parameter	Description	Default
`mag`	magnification	4
`thick`	width of thick line	17.2bp
`thin`	width of thin line	8.3bp
`t`	unit dimen for adjustments	5.5bp
`u`	unit width/height	5.5bp

Table 2: Supplementary parameters used.

Parameter	Description	Default
o_cor	overshoot correction	.5u
lbearing	left bearing	2u
rbearing	right bearing	1u
ascent	distance from baseline of character to top edge	10.4u
dscent	distance from baseline of character to bottom edge	0u

ues ranging from 1 to 4, corresponding to the aforementioned variants: serif, sans-serif, serif thin, and sans-serif thin, respectively. Leveraging the GNU tools within our workflow, passing the value of `width_angle` to the build process is an effortless task. This approach also has the advantage of easily observing the successive outputs of different variants as we construct characters.

We strongly believed that utilizing predefined variables for width and angles relating to different directions, based on the cardinal directions of north, north-east, east, south-east, south, south-west, west and north-west, would greatly enhance comprehensibility and ease of use. These variables would also have fixed values assigned to them. To ensure clarity and consistency, width variables will be prefixed with `w_`, followed by one or two characters indicating the respective direction. Similarly, angle variables will be prefixed with `a_`, followed by the same characters indicating the direction. It's worth noting that the east direction deviates slightly from the expected '`e`' since using '`e`' in the `penpos` command would result in an error. Therefore, the character sequence '`ea`' has been employed in its place.

For a comprehensive list of all the width and angle variables, along with their suggested values, please refer to the following code listing. Please bear in mind that the usage of '`ea`' instead of '`e`' is a unique exception in the provided variables.

For detailed reference, the definitions and corresponding values of variables pertaining to the eight cardinal directions, for each value of the `width_angle` variable, are presented in the subsequent code listings (refer to Listings 1–4). This comprehensive resource offers a valuable point of reference for accessing precise information.

Listing 1: Width and angle variables for `width_angle` value 1 (serif normal).

```
thick:=17.2bp*mag;     % width of thick line
thin:=8.3bp*mag;       % width of thin line
w_cor:=(thick-thin);
w_w=thin;             a_w=180;  % west
w_nw=thin+.25w_cor;   a_nw=135; % north-west
w_n=thin+.5w_cor;     a_n=90;   % north
w_ne=thick-.25w_cor; a_ne=45;   % north-east
w_ea=thick;          a_ea=0;    % east
w_se=thick-.25w_cor; a_se=-45; % south-east
w_s=thick-.5w_cor;   a_s=-90;  % south
w_sw=thin+.25w_cor;  a_sw=-135;% south-west
```

Listing 2: Width and angle variables for `width_angle` value 2 (sans-serif normal).

```
thick:= 1.5u;
thin := 1.5u;
w_cor:=(thick-thin);
t    := 0.0u;
w_w  := thin+.5t;          a_w  := 180;
w_ea := thick+.2t;         a_ea := 0;
w_n  := thin+.2t;          a_n  := 90;
w_sw := thick-.1t;         a_sw := -140;
w_nw := w_sw;              a_nw := 135;
w_ne := w_sw;              a_ne := 45;
w_se := w_ea-1.2t;         a_se := -50;
w_s  := w_n;               a_s  := -90;
```

Listing 3: Width and angle variables for `width_angle` value 3 (serif thin).

```
thick:=8.6bp*mag;
thin:=4.3bp*mag;
w_cor:=(thick-thin);
w_ea:=thick+.2t;           a_ea:=0;
w_w:=thin+.3t;             a_w:=180;
w_n:=thin+.5t;             a_n:=90;
w_sw:=thick-.1t;           a_sw:=-140;
w_nw:=w_sw;                a_nw:=135;
w_ne:=w_sw;                a_ne:=45;
w_se:=w_ea-.6t;            a_se:=-50;
w_s:=w_n;                  a_s:=-90;
```

Listing 4: Width and angle variables for `width_angle` value 4 (sans-serif thin).

```
thick:= .5u;
thin := .5u;
w_cor:=(thick-thin);
t    := 0u;
w_w  := thin+.5t;          a_w  := 180;
w_sw := thick-.1t;         a_sw := -140;
w_ea := thick+.2t;         a_ea := 0;
w_n  := thin+.2t;          a_n  := 90;
w_nw := w_sw;              a_nw := 135;
w_ne := w_sw;              a_ne := 45;
w_se := w_ea-1.2t;         a_se := -50;
w_s  := w_n;               a_s  := -90;
```

3.3 Reusable components

Now let us look into a typical instance, the construction of the consonant character ന, which bears the phonetic resemblance to the initial syllable of the word 'November'. This character is ingeniously brought to life through the deployment of two shape components. Keen observers will note the presence of two distinct lobes—an elegant left lobe and an equally poised right lobe. These lobes find their algebraic expressions in the form of two shapes, known as `c_llobe` and `c_rlobe`, respectively. The nomenclature itself displays an inherent clarity, with the prefix ‘`c`’ symbolizing the component.

The algebraic pursuit of shaping this character involves the harmonious interplay of two fundamental elements. Firstly, we encounter the precise coordinates of each pivotal point that contribute to the formation of the character’s curved contours. Secondly, the width/angle values assigned to each coordinate, accompanied by the stroke command that gracefully connects them, further embellish the visual tapestry. In order to gain a better understanding of this intricate process, the following code unveils the craftsmanship behind its creation.

Listing 5: Listing of the METAPOST source code in the character build file of ന.

```
input mpost-defs;  % MetaPost definitions
input ml-shape-lib;% lib. of shape comps.
input option;  % proofing, width/angle opts
input out;     % PDF/SVG output options

beginfig(34);
  coor_c_llobe (1) (0,0);
  pstroke_c_llobe (1);
  coor_c_rlobe (n1.2) (x1f.r-.5wd2b,0);
  pstroke_c_rlobe (2);
endfig;
end;
```

The files that are evident inputs within the build source, as enumerated in Listing 5, manifest a diverse array of categories. The first file, `mpost-defs.mp`, consists of a select assortment of definitions derived from `plain.mp`, albeit redefined or customized to align with our objectives. The second file, `ml-shape-lib.mp`, is an extensive compendium of shape components with their affiliated macros. Notably, this file internally invokes `ml-glyphs-lib.mp`, which in turn houses the essential definitions of glyphs. The `option.mp` and `out.mp` files help in the build process.

Now, let us examine the macro `coor_c_llobe`, which encompasses the x and y values representing the coordinates of the left lobe. This macro accepts one suffix argument and two expression arguments, namely, `xsh` and `ysh`. These expression arguments serve as containers for the dimensions that dictate the horizontal and vertical shifts of the component when it is positioned within the character construction process.

Listing 6: Listing of the definition of `c_llobe`.

```
def coor_c_llobe (suffix $)(expr xsh,ysh) =
  z$a=(xsh+.55b, ysh+0h);
  z$b=(xsh+.05b, ysh+.5h);
  z$c=(xsh+.8b, ysh+1h); oc$c(-.5);
  z$d=(xsh+1.5b, ysh+.63h);
  z$f=(x$d, ysh+0h);
enddef;

def pstroke_c_llobe (suffix $) =
  penpos$a(w_w-.5t,a_sw);
  penpos$b(w_w+.2t, a_w);
  penpos$c(w_n, a_n);
  penpos$d(w_ea-.1t,a_ea);
  penpos$f(w_ea-.1t,a_ea);
  penstroke subpath (start,stop) of
    (z$a.e .. z$b.e{up} .. {right}z$c.e
      .. z$d.e{down} .. z$f.e);
  penlabels($a,$b,$c,$d,$f);
enddef;
```

In terms of providing the x-coordinate values, we employ the symbol ‘`b`’ to represent breadth, while the y-values are denoted by ‘`h`’, which stands for height. By default, the default values assigned to `b` and `h` are `10u` and `20u` respectively. This flexibility empowers us to modify the values of `b` and `h`, thereby generating condensed or extended variants with ease.

The macro responsible for stroking the paths is named `pstroke_c_llobe`. It encompasses all the `penpos` commands, specifying the width and angle of each coordinate as defined within the corresponding `coor_`⟨*component*⟩ macro. It is worth noting that the width and angle values are expressed in terms of the width/angle variables, as outlined in Listings 1–4, which correspond to the specific font variant being built.

In addition to the `penpos` commands, the path stroking macro also includes one or more (in other cases) `penstroke` commands. These commands connect each coordinate in a sequential manner, employing curves or straight lines as required by the character design. Further, the macro incorporates one or more `penlabels` commands, which facilitate the printing of labels and the left/right edges of the coordinates when generating proofs. It is important to mention that if the value of the `proofing` variable exceeds `2`, the `penlabels` command will also display the angle and width values of each coordinate, providing valuable insights during the debugging phase.

The coordinate and path stroking macros for `c_rlobe` are provided in Listing 7, to allow the reader to examine the implementation.

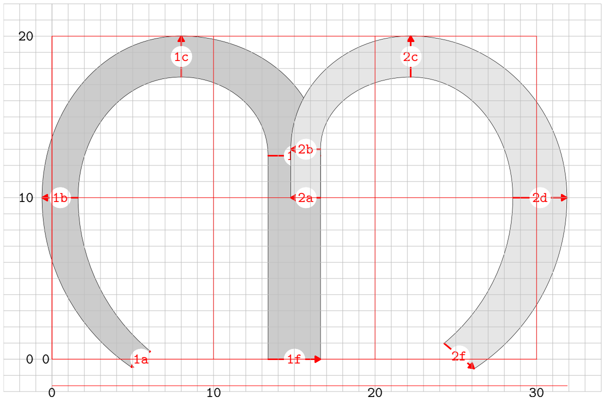


Figure 7: The proof image of the character ന.

Listing 7: Listing of the definition of `c_rlobe`.

```
def coor_c_rlobe (suffix $)(expr xsh,ysh) =
  z$a=(xsh+0b, ysh+.5h);
  z$b=(x$a, .65h);
  z$c=(xsh+.65b, ysh+1h);
  z$d=(xsh+1.45b, ysh+.5h);
  z$f=(xsh+.95b, ysh+0h+.2t);
  y$c:=y$c-.5wd$c;
enddef;

def pstroke_c_rlobe (suffix $) =
  penpos$a(w_sw-.5w_cor,a_w);
  penpos$b(wd$a,a_w);
  penpos$c(w_n,a_n);
  penpos$d(w_ea,a_ea);
  penpos$f(w_s-.1t,a_se+10);
  penstroke subpath (start,stop) of
    (z$a.e{up}
     .. z$b.e{up} .. z$c.e{right}
     .. z$d.e{down} .. z$f.e);
  penlabels($a,$b,$c,$d,$f);
enddef;
```

Figure 7 illustrates the character constructed for the serif normal variant of the font (`wa=1`), utilizing the code presented in Listings 5–7. The components have been visually distinguished using varying shades of gray to aid comprehension.

In a prior section (Section 2, Rationale), we highlighted the fact that the predefined component approach is not exempt from limitations. It is worth noting that there were instances where we encountered the need to adjust the position of coordinates to align with a particular character shape or design. This task proved to be quite challenging, given that the x and y values of the coordinates had been predetermined. Consequently, in order to overcome this obstacle, we undertook the task of redefining the `z` macro [6, p. 277] that assigns values to x and y coordinates. This revised version of the macro now includes (refer to Listing 8) a check for any delta values associated with x or y, to add prior to assigning the respective original values.

Listing 8: The modified definition of `z` macro.

```
vardef z@#=(x@# - if known dx@#: dx@#
    else:0 fi,
    y@# - if known dy@#: dy@# else:0 fi)
enddef;
```

Thus, if there exists a definition for `dx` or `dy` associated with a coordinate `z`, signifying the intended horizontal and vertical shifts respectively, then these shifts will duly be applied to their respective coordinate values prior to final assignment within the pair definition. To illustrate this concept, let us consider the example `dx1b = -2u` (provided in code Listing 9). It is worth noting that the delta values need to be provided just before the occurrence of `coor_c_`⟨*component*⟩.

Listing 9: Example to show the shifting of coordinates.

```
1 beginfig(32);
2   % dx1b=-2u;
3   coor_g_da (1) (0,0);
4   pstroke_g_da (1);
5 endfig;
```

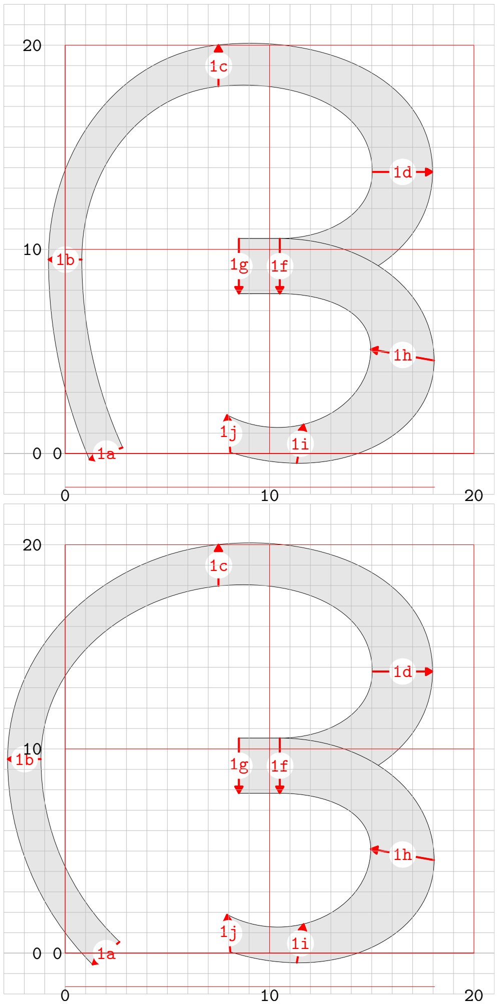
This code generates the consonant character ഡ (using different subroutines than the previous examples). Should we desire a slightly more rounded contour for the left curve, it becomes necessary to adjust the position of the coordinate labelled `1b` towards the left, aligning it with the desired dimensions. This adjustment can be accomplished by employing the delta variable, written as `dx1b = -2u`. It specifies the intended shift of `2u` to the left. It's commented out in line 2 of Listing 9. To gain a visual understanding of the original design alongside the modified version, kindly refer to the images presented in Figure 8.

Similarly, we can utilize the `dy`⟨*coordinate*⟩ approach to shift the vertical position. Nevertheless, the need for precise adjustments extends beyond mere shifts in the x and y directions. At times, it becomes necessary to modify the predetermined angles and widths using the `penpos` commands for individual coordinates. To enable this functionality, we had to redefine the original `penpos` command [6, p. 273], as shown in Listing 10.

Listing 10: Redefined `penpos`.

```
vardef xangle@#(expr xd) =
  (ang@#)=(xd); enddef;
vardef xwidth@#(expr xb) =
  (wd@#)=(xb); enddef;
vardef penpos@#(expr b,d) =
```

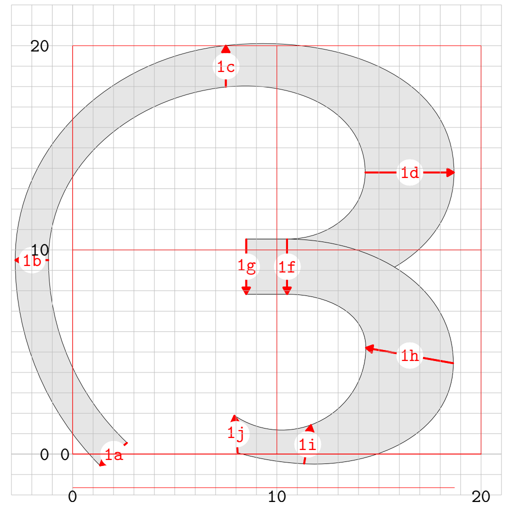
Figure 8: Example showing horizontal shift of a coordinate. The figure at the top is the original character, while the one below shows the midpoint of the left curve shifted by 2u towards the left.




```
  if unknown ang@#: xangle@#(d); fi
  if unknown wd@#:  xwidth@#(b); fi
  (x@#r-x@#l,y@#r-y@#l) =
    (wd@#,0) rotated ang@#;
   x@#=.5(x@#l+x@#r);
   y@#=.5(y@#l+y@#r);
enddef;
```

Let's examine the impact of utilizing these features through a practical example. In the image on the right side of Figure 8, it appears that the angle of coordinate `1a` (the bottom ending of the left-hand stroke) isn't correct as initially set by the command `penpos$a(w_w+.2t, a_sw-20);`. However, it was deemed suitable for the image on the left side of the same figure. To address this, we can make adjustments by inserting the code `ang1a = a_sw;` in the character code; `ang`⟨*coordinate*⟩`=` followed by the value is the syntax of the command.

Figure 9: Revised image of the character after changing the widths and angles.



Listing 11: New source with changed angle and widths.

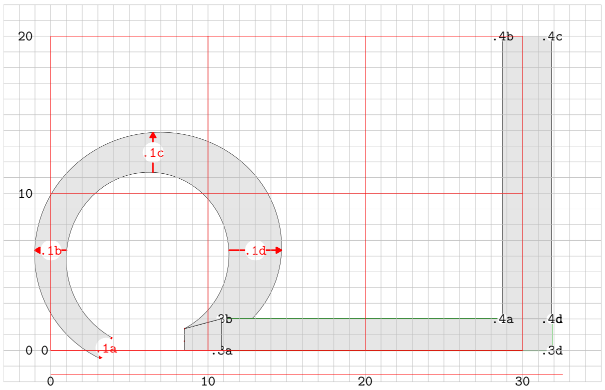
```
beginfig(32);
 dx1b=-2u;   % change x-pos
 ang1a=a_sw; % change angle
 coor_g_da (1) (0,0);
 wd1d=wd1h=w_ea+1t; % change width
 pstroke_g_da (1);
endfig;
```

In a similar fashion, we can also modify the width of any coordinate by using the command `wd`⟨*coordinate*⟩ `=` ⟨*value*⟩. Suppose we wish to alter points `1d` and `1h`. This can be achieved by incorporating the code `wd1d = wd1h = w_ea+1t;`. Essentially, this means that one unit of width will be added to the current width (which was initially defined as `w_ea`).

Please refer to the updated image of the character in Figure 9 and take a moment to compare it with the right image in Figure 8.

There remain a few additional features worthy of explanation, which we shall defer to a subsequent section (refer to Section 3.5). These features give the user additional tools to undertake the task with utmost ease. Among these capabilities are the ability to incise a path at any given point, ascertain the coordinates, angle, and width of said incision point, the redefined `penlabels` command, as well as the utilization of the `find_outline`, `pstroke_stem` and overshoot correction commands.

Figure 10: The consonant character, പ (*pa*).



3.4 Reusable glyphs

The necessity for reusable definitions of glyphs, much like that of reusable components, arose when encountering certain horizontal conjuncts such as ള്ള, ററ, മ്മ, and so forth, where characters are repeated horizontally. Similarly, the need for reusable definitions arose in the case of vertical conjuncts like ണ്ണ, പ്പ, etc., that involve the repetition of the same characters vertically. Such requirements also emerged in cases such as ക്ന, ഷ്ണ, ക്ഷ്മ, and others, where different characters are stacked vertically.

In such circumstances, it is only natural to harness the programmability of METAPOST as a logical progression. This enables the definition of all the glyphs that might undergo repetition, whether in the formation of horizontal or vertical conjuncts, or when glyphs are combined with vowel signs to form ligatures, such as പ്ര, പ്രു, പ്രൂ (phonetically equivalent to *pra*, *pru*, *prū*), derived from the consonant 'പ' (*pa*). Needless to say, the utilization of these reusable definitions greatly expedites the creation of conjunct build files.

Let us now see the typical composition of a glyph definition by carefully examining the source code for the consonant character പ (refer to Figure 10), given in Listing 12. This exploration will shed light on the intricate details that contribute to the formation of this particular character.

Listing 12: The glyph definition of പ from `ml-glyph-lib.mp`.

```
def gl_pa (suffix prx) =
  coor_c_ra_sm (prx.1)(0,0);
  % Lift up end point of ra_sm (1f)
  % and set width relative to its
  % start point (1a)
  y.prx.1f := y.prx.1a.l;
  wd.prx.1f := wd.prx.1a;
  pstroke_c_ra_sm (prx.1);
  stroke_stem (prx.3)
     (x.prx.1f.r+30,0,2.1b,w_w);
  pstroke_stem (prx.4)
     (x.prx.3d-thick,y.prx.3c,thick,1h-w_w);
  pstroke_edge ((x.prx.1f.l,y.prx.3a),
     z.prx.1f.l,z.prx.3b,z.prx.3a);
  reset_xst;
enddef;
```

The definitions of the routines `coor_c_ra_sm` and `pstroke_c_ra_sm` called by `gl_pa` are part of `ml-shape-lib.mp` and are given in Listing 13.

Listing 13: The definition of coordinates, width, angles and path stroke of പ from `ml-shape-lib.mp`.

```
def coor_c_ra_sm  (suffix $)(expr xsh,ysh) =
  z$a=(xsh+.35b, ysh+0); oc$a(.1);
  z$b=(xsh+0b, .5[y$a,y$c]);
  z$c=(.5[x$b,x$d], ysh+.63h);
  z$d=(xsh+1.3b,.5[y$a,y$c]);
  z$f=(xsh+.9b, y$a+.0h);
enddef;

def pstroke_c_ra_sm  (suffix $) =
  penpos$a(w_w-.5t, a_sw+20);
  penpos$b(w_w, a_w);
  penpos$c(w_n, a_n);
  penpos$d(w_ea, a_ea);
  penpos$f(w_s-.5t, a_se);
  penstroke  subpath (start,stop) of
    (z$a.e .. z$b.e .. z$c.e ..
     z$d.e .. z$f.e);
  penlabels($a,$b,$c,$d,$f);
enddef;
```

The commands `pstroke_stem`, `stroke_stem` and `pstroke_edge` and their usage are described in detail in Section 3.5.

The above definitions allow building, fairly easily, a plethora of glyphs (58 in number) listed below, those where പ is an integral part:

പ, പു, പൂ, പൃ, പ്ട, പ്ടു, പ്ടൂ, പ്ത, പ്തു, പ്തൂ, പ്തൃ, പ്ന, പ്ണ, പ്നു, പ്പ, പ്പു, പ്പൂ, പ്പൃ, പ്ഫ, പ്ഫു, പ്ഫൂ, പ്ര, പ്രു, പ്രൂ, പ്ല, പ്ലു, പ്ലൂ, പ്സ, പ്സു, പ്സൂ, പ്സൃ, ഫ്ക, ഫ്കു, ഫ്കൂ, ഫ്പ, ഫ്പു, ഫ്പൂ, ള്പ, ള്പു, ള്പൂ, ള്പൃ, ല്പ, ല്പു, ല്പൂ, ഷ്പ, ഷ്പു, ഷ്പൂ, ഷ്പൃ, സ്പ, സ്പു, സ്പൂ, സ്പൃ, സ്പ്ര, സ്പ്രു, സ്പ്രൂ, ഴ്പ, ഴ്പു, ഴ്പൂ.

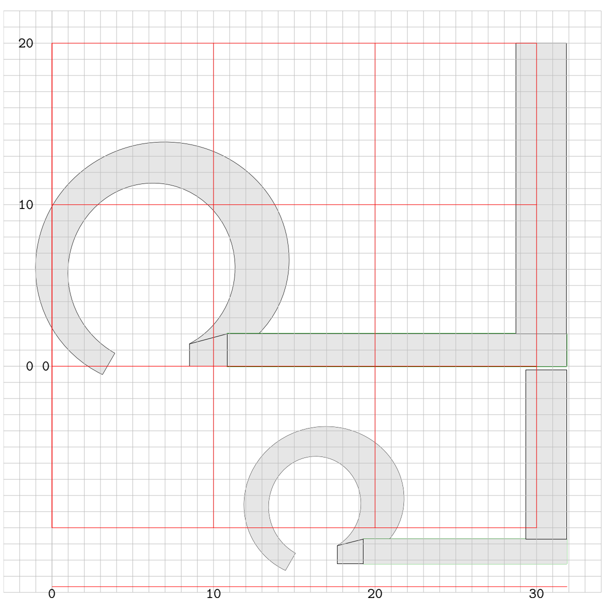
The suffix argument, denoted as `prx`, within the glyph definition of `gl_pa` may pose a puzzling query in the minds of readers, warranting a thorough explanation. It is crucial to comprehend that the suffixes of macros must possess unique identities within a `beginfig ... endfig` environment. Failure to adhere to this requirement will result in an error, halting the processing by METAPOST.

In instances where we need to invoke the same shape functions multiple times, as exemplified in the source code in Listing 14 (its output can be seen in Figure 11), it becomes imperative to ensure the uniqueness of suffixes. To achieve this, we resort to the practice of prefixing the suffixes with additional characters while calling the glyph definitions. (The usage of `vconj` shall be explained in Section 3.5.7 on vertical conjuncts.)

Listing 14: The source listing of `p1p1`, a vertical conjunct പ്പ

```
beginfig(00);
  g1:=image(gl_pa(p));  % first prefix
  vconj:=true; width_angle(wa_n);
  g2:=image(gl_pa(pp)); % different prefix
  g3:=g2 xscaled .6 yscaled .6;
  currentpicture:=g1;
  addto currentpicture also g3
    shifted (xpart (lrcorner g1)
    -xpart(urcorner g3),-(12u+5));
endfig;
```

Figure 11: The vertical conjunct, പ്പ, created by the code in Listing 14.



3.4.1 A word about glyph naming

It's time to provide a brief explanation of the glyph naming convention utilized in our fonts. Instead of using Unicode code points as identifiers for characters, or Adobe glyph list conventions, we have chosen a specialized abbreviated form for both vowels and consonants. This methodology was developed by one of the authors, Hussain, over two decades ago for simplified glyph naming in the fonts he had created, including the most popular, "Rachana".

Vowels are indicated by the prefix `ml_` followed by the corresponding vowel sound in the Latin script. For example, `ml_a` represents the vowel, അ (`0D05`).

Each of the consonant groups, such as velar, palatal, retroflex, dental, bilabial, approximants, sibilants, and others, consist of four stops encompassing all possible values of voicing and aspiration. They are named using one or two representative characters in Latin script (`k` for velar, `ch` for palatal, `t` for retroflex, `th` for dental, `p` biblabial; but each member of the approximants, sibilants and others has been assigned a unique character depending on the sound), followed by a numerical index ranging from 1 to 4. For instance, the first velar consonant ക (`0D15`) is designated as `k1`. As you can infer, the other three (ഖ, ഗ, ഘ) are named `k2`, `k3`, and `k4` respectively. Certain vowel signs and consonants conjoin with many base characters; these are also named appropriately. Table 3 provides a detailed picture of the naming of consonants. It is interesting to note that this naming convention suits most Indic scripts.

Table 3: Table showing all the consonants, vowel signs and their glyph names.

	Voiceless		Voiced		
	Unasp.	Asp.	Unasp.	Asp.	Nasal
velar	k1 ക	k2 ഖ	k3 ഗ	k4 ഘ	ng ങ
palatal	ch1 ച	ch2 ഛ	ch3 ജ	ch4 ഝ	nj ഞ
retroflex	t1 ട	t2 ഠ	t3 ഡ	t4 ഢ	nh ണ
dental	th1 ത	th2 ഥ	th3 ദ	th4 ധ	n1 ന
biblabial	p1 പ	p2 ഫ	p3 ബ	p4 ഭ	m1 മ
approximants	y1 യ	r3 ര	l3 ല	v1 വ	
sibilants	z1 ശ	sh ഷ	s1 സ	h1 ഹ	
others	lh ള	zh ഴ	rh റ		
vowel/ consonant signs	u1 ു	u2 ൂ	y2 ്യ	r4 ്ര	l3 ്ല
	v2 ്വ				

Unasp. = unaspirated; Asp. = aspirated

This convention allows for convenient usage of these intuitive consonant names when constructing conjuncts and ligatures, eliminating the need for lengthy and less user-friendly combinations of

Listing 15: Source listing of the redefined `penlabels`.

```
vardef penlabels@#(text t) =
 if proofing > 1:
   forsuffixes $$=l,r: forsuffixes $=t:
     if known z$:
       interim linecap:=rounded;
       interim ahlength:=8bp;
       interim ahangle:=60;
       drawarrow z$.l -- z$.r
        withcolor red;
       drawdot (x$,y$) withpen pencircle
        scaled 5mm withcolor white;
       defaultscale:=.75;
       s_len:=length(str$); st_idx:=s_len-3;
       makelabel@#(substring(st_idx,s_len)of
        (str$),(x$,y$)) withcolor red;
      if proofing > 2:
       label(decimal(wd$),z$-(0,1u))
        withcolor .5white;
       label(decimal(ang$),z$-(0,1.5u))
        withcolor .5white;
      fi
    fi
   endfor
  endfor
 fi
enddef;
```

code points. Undoubtedly, the glyph name of `k1th1r3` for ക്ത്ര (pronounced like *kthra*) is much easier to remember than the cryptic hex sequence `0D05 0D4D 0D24 0D4D 0D31`.

3.5 Additional features

The supplementary features elucidated in this section are not absolutely imperative for carrying out the font creation process. Nonetheless, they enhance the workflow by equipping users with additional tools that contribute to making their lives a tad more convenient.

3.5.1 Redefined `penlabels` command

The command `penlabels` [6, pp. 36, 274] has been redefined (see Listing 15):

1. to display no labels if `proofing < 2`.
2. if `proofing = 2`, to display the labels in a white circle (since the path is filled with gray color in proof mode) and display a red arrow, the head of which points to the right edge which provides an indication of the angle visually. This is in lieu of the default method of displaying `l` and `r` labels in black.
3. to display the widths and angle of the coordinates if `proofing > 2`, which is handy in certain debugging situations.

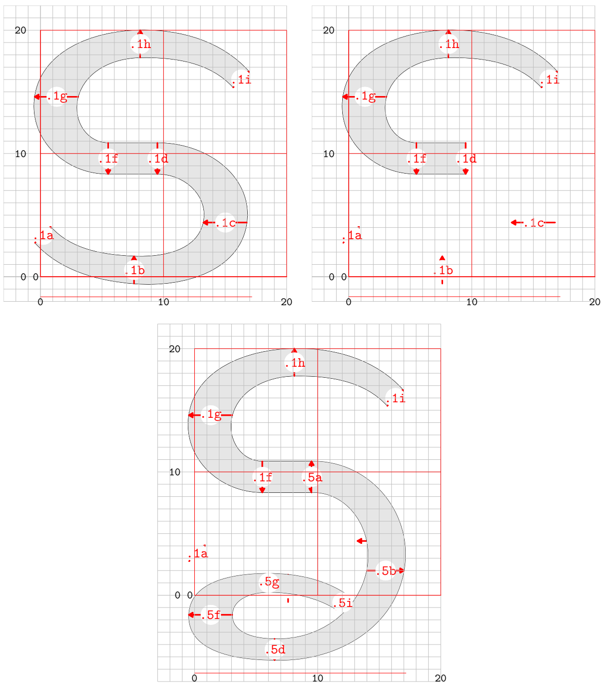
3.5.2 Slicing the path — `subpath`

The `subpath` command [6, p. 133] has been extensively utilized in the `penstroke` macros, as it provides a convenient means to slice the path at arbitrary locations without affecting the path's flow or curvature, even when the cut point happens to intersect a curve. Most `penstroke` commands, if required, include the `subpath` command with `(start, stop)` variables as its arguments, defaulting to `0` and `infinity` respectively. This allows users to modify these variables in the build file as per the dictates of the shape of the glyph. The code, as shown in Listing 16, illustrates how the start and stop points of two consonants, `t1` (ട) and `th3` (ദ), have been applied to the cut just after the start of `t1` and towards the end of `th3` to create derivative glyphs of `t1r1` (ട്ര) and `th3r1` (ദ്ര) respectively.

Listing 16: Application of `subpath`.

```
%%% glyph def of t1 %%%
def gl_Ta =
  coor_g_ta (t1.1) (0,0);
  reset_cut; start:=xstt; % set start of subpath
  pstroke_g_ta (t1.1);    % strokes from 'start'
  reset_xst;
enddef;
%%% glyph def of t1r1  %%%
def gl_TR =
  xstt:=3; gl_Ta; % draw curve from 4th point
  ang_cor:=-5;
  coor_c_prkar (t1.5) (x.t1.1f,3u);
  start:=0;
  wd.t1.5a=wd.t1.1f;
  x.t1.5f:=x.t1.1g;
  x.t1.5a:=x.t1.1d;
  y.t1.5a:=y.t1.1d;
  wd.t1.5f:=wd.t1.1g;
  pstroke_c_prkar (t1.5);
  reset_cut;
enddef;
%%% glyph def of th3  %%%
def gl_da =
  coor_g_da (th3.1) (0,0);
  stop:=xstp; % stop last part of curve at 'xstp'
  pstroke_g_da (th3.1);
  reset_xst;
enddef;
%%% glyph def of th3r1  %%%
def gl_dR =
  dy.th3.1f=-2u;
  coor_g_da (th3.1) (0,0);
  stop:=1; % stop curve at 2nd point
           % of last part
```

Figure 12: Illustration of `subpath` operation. The image at left is the consonant ട, the right-hand image is after slicing at fourth coordinate is applied, and the one below is after appending the vowel sign to the sliced character.



```
pstroke_g_da (th3.1);
coor_c_krkar (th3.2)
  (x.th3.1f-1u,0);
y.th3.2a:=y.th3.1f;
wd.th3.2a=wd.th3.1g;
stop:=infinity;
pstroke_c_krkar (th3.2);
enddef;
```

The three images in Figure 12 illustrate the `subpath` operation with respect to the consonant ട and the second set in Figure 13 is that of ദ.

3.5.3 Overshoot correction

The overshoot correction was previously done using a dimension variable `oc` which has the default value of `0.5u`. Since in serif versions, the characters have different stroke widths at top or bottom owing to the different angles of curves, and the `oc` variable is insufficient to manage overshoot corrections in this situation. However, it was felt that addition of a half stroke width at the bottom and subtraction of a half stroke width at the top will be the ideal solution for it. The images in Figures 14 and 15 illustrate the state of overshoot in pre- and post-application scenario respectively.

Figure 13: Illustration of `subpath` operation (continued). The image at the left is the consonant ദ, the middle image is sliced at the beginning of last lobe and the one below is after appending the vowel sign to the sliced character.

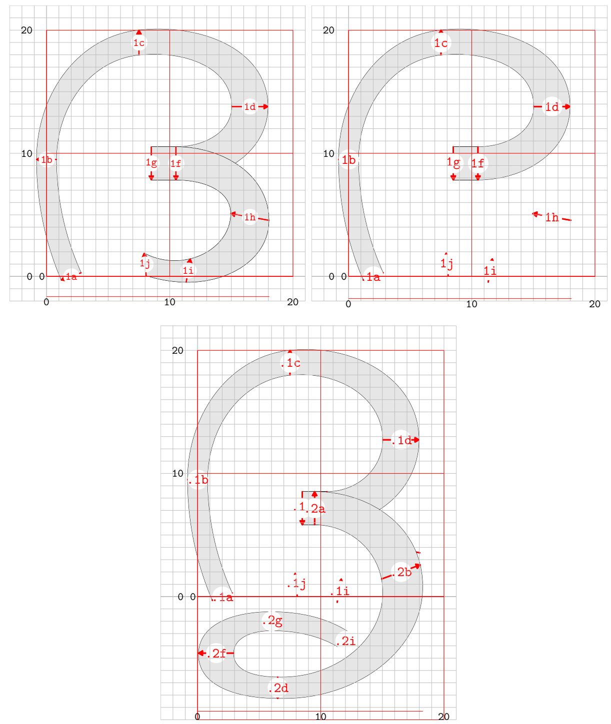
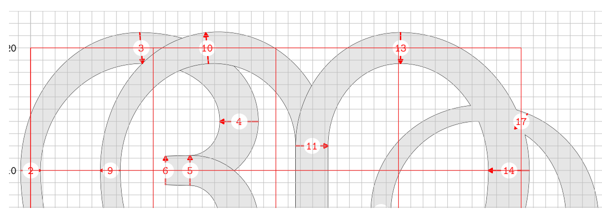


Figure 14: Overshot curves at the top of serif version of the vowel ആ.



The syntax of the command is:

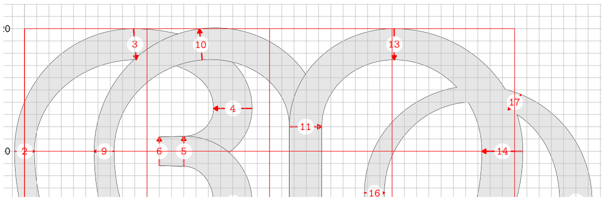
`oc`⟨*coordinate suffix*⟩ `(`⟨*corr value*⟩`)`

A few usage examples are provided below, for situations of both directly coded coordinates like `z1`, `z2`, `z3`, ... and `$`-suffixed situations like `z$a`, `z$b`, `z$c`, ... The correction line is provided just after the coordinate definition of the point needing correction.

```
oc1 (-0.5);
oc5 (0.5);
oc$a(-0.5);
oc$d(0.5);
```

Any arbitrary value can be given as the argument.

Figure 15: Corrected overshot curves at the top of serif version of the vowel ആ.



Listing 17: Definition and usage of `pstroke_stem`.

```
%% definition:
def pstroke_stem  (suffix $)
  (expr xsh,ysh,width,height) =
  x$a=x$b=xsh+0b; x$c=x$d=xsh+width;
  y$a=y$d=ysh+0h; y$b=y$c=ysh+height;
  filldraw z$a -- z$b --
    z$c -- z$d --cycle withcolor gcolor;
  if proofing>0:
    draw z$a -- z$b -- z$c --
      z$d --cycle withcolor black; fi
  labels($a,$b,$c,$d);
enddef;
%% usage:
%  pstroke_stem (<suffix>)
%        (<h-shift>, <v-shift>,
%         <width>, <height>);
```

3.5.4 The `pstroke_stem` macro

The horizontal and vertical stems that form part of some characters (*e.g.*, വ, ഖ, പ, ഷ, ഥ, ല, ഘ, ...) are drawn using the function `pstroke_stem`; its usage is shown in Listing 17.

After one argument for the suffix, the function requires four expression arguments: horizontal and vertical shifts, width, and height of the stem. If the height is greater than the width, the stroke becomes a vertical stem.

Readers are encouraged to examine lines 9–12 of Listing 12, where the usage of `pstroke_stem` is evident. Instead of `pstroke_stem`, you can see `stroke_stem` in line 10. Both have the same functionality, except that the latter invokes `outline` mode which is explained later (Section 3.5.10).

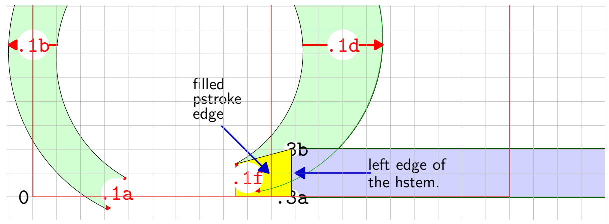
The image in Figure 10 demonstrates the effect of the aforementioned code.

3.5.5 The `pstroke_edge` macro

There exists a typographical nuance of reducing the height of an edge of a horizontal stem when it joins with a curved stroke as seen in Figure 16.

The `pstroke_edge` command draws a cyclic path connecting the left end coordinates of the hor-

Figure 16: Example of `pstroke_edge` when the left edge of a horizontal stem joins with the rounded part of the character. The path in light green color is the rounded path; the hstem is colored light blue while the segment filled with yellow color is the `pstroke_edge`.



izontal stem (points `3a` and `3b` in the figure) with the left and right edges of end point of the curve (`1f`) and fills it. The macro definition and usage are provided in Listing 18.

Listing 18: Definition and usage of `pstroke_edge`.

```
%% definition:
def pstroke_edge (expr ll,ul,ur,lr) =
 filldraw ll -- ul -- ur -- lr --
   cycle withcolor gcolor;
  if proofing>0:
   draw ll -- ul -- ur -- lr --
   cycle withcolor black; fi
enddef;
%% usage example:
%% lines 13, 14 of Listing 12
 pstroke_edge (
   (x.prx.1f.l, y.prx.3a),
    z.prx.1f.l,
    z.prx.3b,
    z.prx.3a
 );
```

The four coordinates needed for `pstroke_edge` can be provided starting from any point, as long as the coordinates are sequentially in cyclic order, no matter clockwise or anticlockwise.

3.5.6 `DocGrid` and `PrintGrid`

The `DocGrid` macro overlays a grid on top of the glyph image in `proofmode` (`proofing > 0`). It is an extended form of Knuth's `makegrid` macro, explained in [6, p. 275]. Listing 19 provides the source code of the macro.

Listing 19: The source code of `DocGrid`.

```
def DocGrid (expr w,h) =
 if proofing > 0:
   begingroup
     defaultscale := 1.1;
     pickup pencircle scaled minor_rulewidth;
     rulecolor:=minor_rulecolor;
     bm=(ypart(llcorner pp)-1u);
```

```
    makegrid(0,for i=u-4u step u
      until w+3u: , i endfor)
     (0,for i=0 step u
      until h+2u+1: , i endfor)
    makegrid(0,for i=u-4u step u
     until w+3u:, i endfor)
     (0,for i=0 step -u
      until bm-1u: , i endfor)
  pickup pencircle scaled major_rulewidth;
 rulecolor:= major_rulecolor;
 makegrid(0,for i=0 step 10u until w+2u+1u: ,
  i endfor)
 (0,for i=0 step 10u until h+2u+1: ,
  i endfor);
 makegrid(0,for i=0 step 10u
  until w+2u+1u: , i endfor)
 (0,for i=0 step -10u
  until bm-1u: , i endfor);
 draw (0,bm)--(w,bm) withcolor rulecolor;
 makelabel.lft("0",(0u,0u));
 makelabel.bot("0",(0u,bm));
 makelabel.bot("10",(10u,bm));
 makelabel.bot("20",(20u,bm));
 if (w+2u) >= 30u :
  makelabel.bot("30",(30u,bm)); fi
 if (w+2u) >= 40u :
  makelabel.bot("40",(40u,bm)); fi
 if (w+2u) >= 50u :
  makelabel.bot("50",(50u,bm)); fi
 if (w+2u) >= 60u :
  makelabel.bot("60",(60u,bm)); fi
 if (w+2u) >= 70u :
  makelabel.bot("70",(70u,bm)); fi
 makelabel.lft("0",(-1u,0u));
 makelabel.lft("10",(-1u,10u));
 makelabel.lft("20",(-1u,20u));
 endgroup;
fi;
enddef;
```

The DocGrid macro requires the width and height of the character as its arguments and adds 2u space around the bounding box of the character before overlaying with the grid. It is invoked by the PrintGrid function though the endfig hook, extra_endfig, as shown in Listing 20.

Listing 20: The source code of PrintGrid.

```
def PrintGrid =
  picture pp; pp:=currentpicture;
  pw = xpart(urcorner pp);
  ph = ypart(urcorner pp);
  DocGrid(pw,ph);
enddef;

extra_endfig:="if proofing = 0:
  add_space_around; else: PrintGrid; fi";
```

3.5.7 The vertical conjuncts

The process of generating vertical conjuncts, where two characters are stacked on top of each other, was mentioned in Section 3.4, Reusable glyphs. The corresponding source code in Listing 14 demonstrated how this is achieved. You may notice that the below-base character is always scaled down to ensure typographic appeal and to limit the overall depth of the glyph to a reasonable level. However, this scaling action has the unintended consequence of reducing the stroke width in the bottom character.

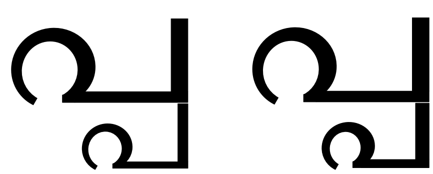
To address the limitation of width reduction, a boolean variable called vconj is introduced. When set to true, it automatically increases the width of the thick and thin lines by a factor of 1.35. Since all other stroke widths are derived from these two fundamental dimensions, the overall width dimensions are adjusted proportionately and accurately.

Listing 21: Change of the widths thick and thin depending on the state of the boolean, vconj.

```
wa:=1;
if vconj:
  thick:=(1.35*17.2bp)*mag; % width of thick line
  thin:=(1.35*8.3bp)*mag;   % width of thin line
else:
  thick:=17.2bp*mag;        % width of thick line
  thin:=8.3bp*mag;          % width of thin line
fi
```

The images in Figure 17 show the difference in stroke widths of the below-base character with different states of the vconj boolean.

Figure 17: The effect of the boolean vconj on the stroke width of the below-base character. The image on the left is when vconj is false; observe that the widths of strokes of the below-base character are thinner than its counterpart on the right, with vconj true.



3.5.8 The consonant doubling macro

Some consonants, such as ച്ച, ബ്ബ, യ്യ, വ്വ (phonetically similar to *chcha*, *bba*, *yya*, *vva*), behave in a particular fashion when conjuncts with the same consonants are created. This is different from others like പ്പ, ണ്ണ, ഗ്ഗ (*ppa*, *ṇṇa* as in ru*nn*ing, *gga*), etc., when forming vertical conjuncts with its own copy of the below-base character and മ്മ, ത്ത, ള്ള (*mma*, *ththa*, *ḷḷa* as in cu*ll*ing), etc., where conjuncts are formed with their own copy of the post-base character packed horizontally.

Since the shape of the bottom construct is an ideal candidate for a component, we devised one, called `c_cons_dbl`, the source of which is provided in Listing 22.

Listing 22: Consonant doubling macro, `c_cons_dbl`.

```
def c_cons_dbl (suffix $)(expr xsh,ysh,wid,hgt) =
  pstroke_stem ($) (xsh+0b,ysh+0h,-thick,-hgt);
  dx$a.c=-.15b;
  stroke_stem ($a)(x$b,y$b-thin,-wid,thin);
  if not outln_i: ypenstroke stem; fi
  z$aa=(x$a-.65wid,y$a); penpos$aa(w_n,a_w);
  z$ab=(x$a.d-.2b,y$a.d+.5wd$ab);
  penpos$ab(.6thin,a_n);
  penstroke z$aa.e {dir 263} .. z$ab.e;
  penlabels($aa,$ab);
  pstroke_edge((x$ab.r,y$a.d),z$ab.r,z$a.c,z$a.d);
enddef;
```

Figure 18: The consonant doubling macro in action.

ച്ച ഞ്ഞ യ്യ വ്വ

The macro requires four expression arguments of horizontal shift, vertical shift, width and height, after the suffix argument. A usage example showing the source of the conjunct ച്ച is provided in Listing 23.

Listing 23: Usage of the consonant doubling macro.

```
beginfig(20);
  coor_c_ch_lt (1) (0u,0u);
  pstroke_c_ch_lt (1);
  pstroke_stem (2)(0,0,3b,thin);
  pstroke_stem (3)(x2d,0,thick,1h);
  c_cons_dbl (4)(x3d,y3a,1.8b,.4h);
endfig;
```

3.5.9 The vowel signs

Among all the vowel signs, four of them — ◌ു, ◌ൂ, ◌ൃ, ◌ൄ — exhibit the tendency to join with consonants to form conjuncts. Another speciality is that each of the first two — ◌ു and ◌ൂ — has four different forms depending on the shape and other characteristics of the conjoining consonant. A few examples below illustrate the diverse conjunct formation with the sign ◌ു:

(i) കു, രു (*ku*, *ru*);
(ii) ഗു, ജു, തു, … (*gu*, *ju*, *thu*);
(iii) നു, ണു, ന്നു, … (*nu*, *ṇu*, *nnu*) and
(iv) ദു, പു, ബു, മു, … (*du*, *pu*, *bu*, *mu*).

Similarly, the longer form ◌ൂ creates four different conjuncts that correspond to the shorter forms cited in the previous list:

(i) ജൂ, രൂ (*jū*, *rū*);
(ii) കൂ, ഗൂ, തൂ, … (*kū*, *gū*, *thū*);
(iii) നൂ, ണൂ, ന്നൂ, … (*nū*, *ṇū*, *nnū*) and
(iv) ദൂ, പൂ, ബൂ, മൂ, … (*dū*, *pū*, *bū*, *mū*).

However, the other two vowel signs — ◌ൃ, ◌ൄ — do not tend to create different kinds of conjuncts. The first assumes the uniform shape of കൃ, ഗൃ, പൃ, ദൃ, … while the conjuncts of the latter have the shape of കൄ, ഗൄ, പൄ, ദൄ, … There is yet another consonant, ര, that conjoins with base characters to make the 'reph' form as ക്ര, ഗ്ര, പ്ര, ദ്ര, … that is defined using `make_reph` macro (see Table 4).

We observed that the majority of the conjuncts with the first two vowel signs tend to form rounded variants of ◌ു and ◌ൂ (see the examples in item iv of the above two lists); hence, two macros have been designed so that the corresponding conjuncts are created via simple calling of one of these macros augmented with appropriate x, y coordinate values at which to attach. The two examples of the conjuncts പു and പൂ provided in Figure 19 illustrate the process. The corresponding METAPOST sources are in Listing 24.

Figure 19: Formation of conjuncts with rounded vowel signs aligned with the bottom of a horizontal stem.

പു പൂ

Listing 24: The conjuncts with rounded vowel forms.

```
%%% പു %%%
def gl_pu (suffix prx) =
  outln:=true;
  gl_pa(prx);
  make_stem_u (21) (x.prx.3d-.5wd21c,
    y.prx.3c-.75wd21b);
enddef;
%%% പൂ %%%
def gl_puu (suffix prx) =
  outln:=true;
  gl_pa(prx);
  make_stem_uu (21) (x.prx.3d-(x21c.r-x21b),
    y.prx.3c-.5wd21b);
```

Two macros, `make_stem_u` and `make_stem_uu` are used to align the ◌ു and ◌ൂ respectively with the horizontal stem of ഫ. One may also notice a boolean `outln` has been set true to improve the alignment process (this will be explained in detail in Section 3.5.10 on `outline` mode). The macro requires two arguments, the horizontal and vertical coordinates, where the rounded object will align with the stem. The sources in Listing 25 will amplify this further.

Listing 25: The conjuncts with rounded vowel forms conjoining with horizontal stems of consonants.

```
%%% ◌ു-sign %%%
def make_stem_u (suffix $) (expr xsh,ysh) =
  coor_vl_round_u_alt ($) (xsh,ysh);
  pstroke_vl_round_u_alt (21);
  if not noreverse: stem:= reverse stem; fi
  find_outlines(rmpath,stem)(P);
  for i=1 upto P.num: ypenstroke P[i]; endfor
enddef;
%%% ◌ൂ-sign %%%
def make_stem_uu (suffix $) (expr xsh,ysh) =
  coor_vl_round_uu_alt ($) (xsh,ysh);
  pstroke_vl_round_uu_alt (21);
  if not noreverse:stem:= reverse stem; fi
  find_outlines(rmpath,stem)(P);
  for i=1 upto P.num: ypenstroke P[i]; endfor
enddef;
```

As you might surmise, there are also variant forms of the macros to accommodate rounded bottom curves where the vowel sign is expected to align. The source code is provided in Listing 26, and the corresponding output in Figure 20.

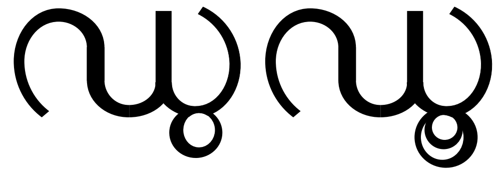
Listing 26: The conjuncts with rounded vowel forms conjoin with bottom curves of consonants.

```
%%% ഡു  %%%
def gl_Du (suffix prx) =
  outln:=true;
  gl_Da (prx);
  make_round_u (21) (x.prx.4a+20+(x21c-x21b),
                    y.prx.4a.r-.75wd21b);
enddef;
%%%  ഡൂ  %%%
def gl_Duu (suffix prx) =
  outln:=true;
  gl_Da (prx);
  make_round_uu (21) (x.prx.4a,y.prx.4a+.1wd21b);
enddef;
```

The other forms of these two vowels are limited to a very few consonants and therefore created individually by slicing the paths at appropriate locations and adding necessary components from the `ml-shape-lib.mp` library.

A variety of macros, listed in Table 4, defined in `ml-vlsigns-lib.mp`, can be used to create conjuncts with different vowels depending on the shape of the consonant and the final form of the conjunct.

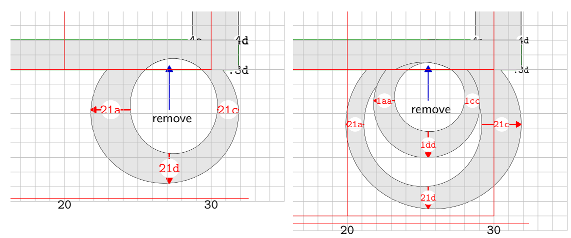
Figure 20: Formation of conjuncts with rounded vowel signs aligned with the bottom of a curve.



3.5.10 `outline` mode

The rounded forms of the vowel signs ◌ു and ◌ൂ present the unusual challenge of excising the portions not encompassed by their curved structures, when juxtaposed on a horizontal stem or rounded path. This typographical intricacy is meticulously adhered to by all the fonts crafted and published by the Rachana Institute of Typography. Figure 21 shows two illustrative images, demonstrating this requisite nuance, ensuring harmonious integration of the rounded vowel signs within the textual fabric.

Figure 21: Example images showcasing the removal of uncovered parts from ◌ു and ◌ൂ.



Implementing this particular requirement in METAPOST presents itself as a formidable undertaking. Happily, however, the bundled `plain_ex.mp` library accompanying the METATYPE1 package [4] emerged as a veritable fairy godmother to navigate this very challenge. Thus we will share a snippet from the self-documented code base of `plain_ex.mp`, perfectly suited for the readers seeking enlightenment in this matter.

> The problem can be stated as follows: two paths are given (precisely: expressions of type **path**); assume that the positively directed (anti-clockwise) path accomplishes filling, and negatively directed (clockwise) — erasing; the task is to find the outline of the resulting (visible) figure. Such a task is known as "removing overlaps" which seems too narrow for such a complex operation. Actually, the basic macro of that part, i.e., `find_outlines`, accomplishes set-theory operations: sum, difference and product, depending on the turning number of the input paths. The illustration below demonstrates the results

Table 4: List of macros, glyphs in which used, and the glyphs in Malayalam script.

Macro	Glyph ID	ML glyph
`c_thrkar`	`h1r1, k1k1r1, k3r1, ch2r1, th1r1, th1th1r1, k1th1r1, n1r1, n1n1r1, n1th1r1, z1r1, z1ch2r1`	ഹൃ, ക്കൃ, ഗൃ, ഛൃ, തൃ, ത്തൃ, ക്തൃ, നൃ, ന്നൃ, ന്തൃ, ശൃ, ശ്ഛൃ
`c_krkar`	`n1th3r1, ngk1r1, p4r1, r3r1, th1p4r1, k1r1, k3th3r1, th3r1, th3th3r1`	ങ്കൃ, ഭൃ, ഋ, ന്ദൃ, കൃ, ഗ്ദൃ, ദൃ, ദ്ദൃ
`c_prkar`	`ch3r1, ch3ch3r1, h1m1r1, k1shr1, k2r1, k3k4r1, k4r1, l3r1, m1r1, m1p1r1, n1m1r1, n1th3r1, njch1r1, njch2r1, p1r1, p3r1, shr1, s1th2r1, t1r1, th1m1r1, th1th2r1, th1s1r1, v1r1`	ജൃ, ജ്ജൃ, ഹ്മൃ, ക്ഷൃ, ഖൃ, ഗ്ഘൃ, ഘൃ, ലൃ, മൃ, മ്പൃ, ന്ദൃ, ഞ്ചൃ, ഞ്ഛൃ, പൃ, ബൃ, ഷൃ, സ്ഥൃ, ടൃ, ത്മൃ, ത്ഥൃ, ത്സൃ, വൃ
`c_srkar`	`ch4r1, k3th3th4r1, n1th4r1, nht3r1, nht4r1, s1r1, t3r1, t4r1, th1s1r1, th3th4r1, th4r1, y1r1`	ഝൃ, ഗ്ദ്ധൃ, ന്ധൃ, ണ്ഡൃ, ണ്ഢൃ, സൃ, ഡൃ, ഢൃ, ത്സൃ, ദ്ധൃ, ധൃ, യൃ
`u_bot`	`ch2u1, ch3u1, ch3u2, ch3ch3u2, h1u1, h1u2, k1k1u2, k3u1, k3u2, k1th1u1, k1th1u2, n1th1u1, n1th1u2, njch2u1, p4u1, p4u2, th1p4u2, th1u1, th1u2, th1th1u1, th1th1u2, z1u1, z1ch2u1`	ഛു, ജു, ജൂ, ജ്ജൂ, ഹു, ഹൂ, ക്കൂ, ഗു, ഗൂ. ക്തു, ക്തൂ, ന്തു, ന്തൂ, ഞ്ഛു, ഭു, ഭൂ, ത്ഭു, ത്ഭൂ, തു, തൂ, ത്തു, ത്തൂ, ശു, ശ്ഛു
`c_chhuu`	`k1u2, ch2u2, ngk1u2, njch2u2, z1u2`	കൂ, ഛൂ, ങ്കൂ, ഞ്ഛൂ, ശൂ
`c_ku_rt`	`ch3u2, ch3ch3u2, njch3u2, p4u2, th1p4u2`	ജൂ, ജ്ജൂ, ഞ്ജൂ, ഭൂ, ത്ഭൂ
`c_noo_rt`	`h1n1u2, k3n1u2, n1n1u2, n1u2, n1n1u2, nhu2`	ഹ്നൂ, ഗ്നൂ, നൂ, ന്നൂ, ണൂ
`make_reph`	`ch1ch2r3, ch2r3, ch3r3, ch4r3, h1r3, k1k1r3, k1r3, k1t1r3, k1th1r3, k3r3, k3th3th4r3, k4r3, l3p1r3, m1p1r3, m1r3, n1m1r3, n1n1r3, n1r3, n1th1r3, n1th3r3, n1th4r3, nht1r3, nht3r3, p1r3, p1s1r3, p2r3, p3r3, p4r3, r3, s1k1r3, s1p1r3, s1r3, s1s1r3, s1t1r3, s1th1r3, s1rhrhr3, shk1r3, shp1r3, shr3, sht1r3, t1r3, t3r3, t4r3, th1r3, th1s1r3, th1th1r3, th3r3, th3th4r3, th4r3, v1r3, z1r3, z1z1r3`	ച്ഛ്ര, ഛ്ര, ജ്ര, ഝ്ര, ഹ്ര, ക്ക്ര, ക്ര, ക്ട്ര, ക്ത്ര, ഗ്ര, ഗ്ദ്ധ്ര, ഘ്ര, ല്പ്ര, മ്പ്ര, മ്ര, ന്മ്ര, ന്ന്ര, ന്ര, ന്ത്ര, ന്ദ്ര, ന്ധ്ര, ണ്ട്ര, ണ്ഡ്ര, പ്ര, പ്സ്ര, ഫ്ര, ബ്ര, ഭ്ര, ര, സ്ക്ര, സ്പ്ര, സ്ര, സ്സ്ര, സ്ട്ര, സ്ത്ര, സ്റ്റ്ര, ഷ്ക്ര, ഷ്പ്ര, ഷ്ര, ഷ്ട്ര, ട്ര, ഡ്ര, ഢ്ര, ത്ര, ത്സ്ര, ത്ത്ര, ദ്ര, ദ്ധ്ര, ധ്ര, വ്ര, ശ്ര, ശ്ശ്ര

yielded by the macro `find_outlines`. There are four cases since there are four combinations of turning numbers for two "regular" paths. Each case shows the initial situation (left) and the resulting one (right). Filling is omitted, the outline colour shows the turning number: blue — positive, red — negative.

In the realm of typography, where every detail matters, the `plain_ex.mp` library proves itself to be an indispensable ally. Therefore, without a moment's hesitation, the `find_outlines` macros have been unabashedly borrowed in their entirety from the `plain_ex.mp` library. They have been encapsulated within a function and employed to scrutinize the paths whenever the rounded vowel signs align themselves alongside the horizontal stem or the curved lower boundaries of the consonants. This judicious integration ensures that the typographical integrity remains intact. Many of the components that form part of the bottom curves and the `pstroke_stem` macros have been redefined to include code to check the state of `outln` and apply `find_outlines` macros appropriately depending on the state of the boolean. A closer examination of the METAPOST sources listed in Listings 24 through 26 will provide ample illustration of the above.

4 OpenType font generation

METAFONT gives us parametric font design and METAPOST gives us vector format glyphs, SVG being the preferred output format. Contemporary font building tools work with vector glyph outlines; and we have a few free software choices, viz., FontForge, `fontmake`, etc. We did look at METATYPE1 but it is suitable for generating Type 1 fonts only; still, macros such as `find_outlines` from METATYPE1 are made use of. Khaled Hosny has built an OpenType version of Knuth's *Punk* font [2], utilizing the Python scripting abilities of FontForge (we thank Hosny for adding an open license to the build script on our request). We had earlier developed font building tools based on FontForge for Malayalam font development and this was found to be a natural fit.

We thus developed a font build tool in Python, heavily utilizing FontForge Python libraries, driven by a configuration file to (1) import the SVG outlines into glyph slots; (2) assign Unicode codepoints to them; (3) set left/right bearing, width and other properties; (4) set font metadata and (5) generate the final OpenType font formats such as TTF, OTF, WOFF. Thus, the overall development workflow is:

METAPOST ⟶ SVG ⟶ FontForge + scripts ⟶ OTF/TTF/WOFF

Each character is defined in its own METAPOST file following the glyph naming convention, which is then compiled to generate an SVG file in a directory for the variant being generated (`serif-regular`, `serif-thin`, etc.), bearing the same glyph name, for e.g., `ml_a.mp` → `ml_a.svg`. In general, all the glyph names in the METAPOST files, generated SVG files, config files and OpenType layout rules follow the convention explained in Section 3.4.1.

Once the SVG files are generated, the next round of scripts, driven by a configuration file that associates glyph names with its Unicode codepoint and other properties, are run. The Malayalam script has many conjuncts that do not have individual codepoints themselves but are formed by a combination of basic Unicode codepoints. These glyphs do not need a mapping entry for codepoint but may need other properties.

The configuration file also holds the font metadata such as family name, PostScript name, variant name, version, license, etc. Crucially, it should also be possible to adjust the left/right side bearings of all the glyphs. A default bearing value suffices for most; but quite a few need either a smaller or negative value, for e.g. ◌ി (`U+0D3F`, glyph name `i1`) and ◌് (`U+0D4D`, glyph name `xx`) which extends outside the left margin (negative side bearing). In addition, setting the width for non-printable characters like `space` is crucial, as this affects word spacing.

The common `ini` configuration file format is used; and a self-explanatory sample file is provided in Listing 27.

Listing 27: Configuration file for font building.

```
# Metadata
[font]
family=Sayahna
name=Sayahna-Regular
version=0.9.1
ascent=820
descent=180
copyright=Copyright 2021-2023 Rachana
          Institute of Typography
          <info@rachana.org.in>

# SVG, OpenType feature file, Unicode mapping
[source]
glyphdir=svgs-regular/
featurefile=features/sayahna-feature.fea
ucglyphmapfile=tools/rit-ml-uc-glyph.map

# Width of specific glyphs
[width]
space=300

# Default and overridden left, right bearings
[bearing]
default=30,40
i1=-74,30   # negative left bearings
i2=-80,30
r1=-112,30
xx=-57,30
y2=-70,30
y2u1=-70,30
y2u2=-70,30
v2=-40,30
```

The font build tool performs a number of steps:

- Assemble all the SVG format glyphs found in the target directory `glyphdir`.
- Remove overlap of outlines.
- Add additional glyphs/codepoints for space, zero-width joiner/non-joiner (`U+0020`, `U+200D`, `U+200C`), etc. These are used in certain character combinations.
- Adjust side bearings/widths of glyphs as specified in the config file.
- Set metadata.
- Assign the Unicode code points for all base characters, given in a simple mapping file `rit-ml-uc-glyph.map`.
- Apply the OpenType shaping rules for Malayalam [8], given in a file `sayahna-feature.fea`.
- Set kerning values as specified.
- Finally, produce the font in any desired format(s), such as TTF, OTF or WOFF2.

All these steps are fully automated by a few Makefile targets. Thus, if a font developer wishes to make amendments to a glyph, she can make the changes in the METAPOST file and then run a `make font` (or equivalent) command that goes through all the above steps and generates the revised font a few moments later.

5 Availability

All of the source code of the project, including METAPOST libraries, glyph definitions, font build tools, OpenType shaping rules, Unicode mapping files and other configuration files, is available at `gitlab.com/rit-fonts/Sayahna-font` under free software licenses, such as the LPPL and the OFL.

6 About the Rachana Institute of Typography

The Rachana Institute of Typography (RIT) is a not-for-profit digital foundry in Kerala, India. To date, it has released ten families of Indic script fonts in TrueType and WOFF2 formats under the terms of the Open Font License, with more planned.

The following are the technical objectives underlying the founding of the RIT:

1. To resurrect the original script of Malayalam, which was diluted by dropping hundreds of conjuncts and ligatures when a reformed script for the language was introduced by the government of Kerala to fit the typewriter keyboards.
2. To develop fonts that completely adhere to the definitive character set prevalent through the ages.
3. To completely revamp the OpenType layout (i.e., the GSUB table) of the RIT fonts to uphold the right spirit of a comprehensive character set and to replace all previous modifications.
4. To redesign all the glyphs in the flagship font, RIT Rachana.
5. To freshly redesign RIT Rachana Italic, Bold and Bold Italic and to generate respective font variants.
6. To encourage use of code-driven font development models by employing character description languages such as METAFONT and METAPOST.

The RIT Rachana font is used throughout this article. For samples, font charts, usage options, etc., see the documentation of the `rit-fonts` package: `ctan.org/pkg/rit-fonts`.

References

[1] J. Hobby, et al. *MetaPost: A Users Manual*. `tug.org/metapost`

[2] K. Hosny. The Punk Nova font. 2010. `github.com/aliftype/punk-otf`

[3] K.H. Hussain, R. Chitrajakumar, et al. The Rachana font family. 2023. `gitlab.com/rit-fonts/RIT-Rachana`

[4] B. Jackowski, J.M. Nowacki, P. Strzelczyk. Programming PostScript Type 1 fonts using METATYPE1: Auditing, enhancing, creating. *TUGboat* 24(3), 2003. Proceedings of EuroTEX 2003. `tug.org/TUGboat/tb24-3/jackowski.pdf`

[5] D.E. Knuth. *The TEXbook*, vol. A of *Computers & Typesetting*. American Mathematical Society and Addison Wesley, Reading, Massachusetts, 1986.

[6] D.E. Knuth. *The METAFONTbook*, vol. C of *Computers & Typesetting*. American Mathematical Society and Addison-Wesley, Reading, Massachusetts, 1986.

[7] Rachana Institute of Typography. Malayalam glyphs, components and sectors. 2022. `rachana.org.in/docs/rit-comp-list.pdf`

[8] K.V. Rajeesh. Malayalam OpenType shaping rules. 2020. `gitlab.com/rit-fonts/malayalam-shaping/`

[9] Unicode. Malayalam: 0D00–0D7F. `unicode.org/versions/latest/ch12.pdf` `unicode.org/charts/PDF/U0D00.pdf`

[10] Wikipedia. Brahmic scripts. `wikipedia.org/wiki/Brahmic_scripts`

[11] Wikipedia. Languages of India. `wikipedia.org/wiki/Languages_of_India`

◇ C. V. Radhakrishnan
River Valley Technologies, River Valley Campus, Malayinkeezh
Trivandrum 695571, India
`cvr (at) river-valley (dot) org`
`http://river-valley.com`
ORCID 0000-0001-7511-2910

◇ K. V. Rajeesh
Rachana Institute of Typography
`rajeesh (at) rachana (dot) org (dot) in`
`https://rachana.org.in/`

◇ K. H. Hussain
Rachana Institute of Typography, Jagathy, Trivandrum 695014, Kerala, India
`hussain (at) rachana (dot) org (dot) in`
`https://rachana.org.in`

TUG 2023 abstracts

Editor's note: Links to videos and other information are posted at tug.org/tug2023.

– – ∗ – –

Patrick Gundlach

News from boxes and glue: How do the TeX algorithms help in developing a new typesetting engine?

In this presentation I will talk about the experience of the last two years with boxes and glue. The library has not yet reached its final state, but a lot has already been typeset with it. I will show what kind of experiences I have made with the TeX algorithms, which data structures are suitable for text typesetting and how PDF specialties like interaction and accessibility can be integrated.

About boxes and glue: boxes and glue is a library written in the Go programming language that includes many of TeX's algorithms, such as the optimum fit paragraph breaking algorithm, the hyphenation algorithm, and the basic structure with nodes and node lists to assemble boxes. It was originally written as a replacement for LuaTeX to create documents with the speedata Publisher.

Island of TeX

The Island of TeX 2023 — sailing the smooth seas of ideas

The Island of TeX has always valued community over development pace. This year, we are proud that we could convince our inner sloths to produce a long-awaited new `albatross` release and a new website for our community. On the technical side, we improved our build infrastructure and started welcoming TeX packages. But in the end, this year was primarily about collecting ideas so stay tuned for our talk and call for action.

Oliver Kopp

JabRef as BibTeX-based literature management software

JabRef is literature management software completely based on the BibTeX format. This talk provides an overview of JabRef by first introducing the basic concept of JabRef. After that, highlights of JabRef will be demonstrated: Integrated web search, grouping of entries, import and export of other formats, and the quality assurance of entries. The integration of PDFs will be demonstrated: Both the linking of PDFs and the integration of BibTeX data into PDFs using XMP metadata.

Eberhard W. Lisse

Introduction to Typst

typst is a new markup-based typesetting system that is designed to be as powerful as LaTeX while being much easier to learn and use. It flows from a Master's thesis at the Technical University Berlin, is written in Rust, and has a domain-specific language that is much easier to master than TeX or LaTeX. It produces quite reasonable output, and especially for shorter documents it is extremely fast, though it remains a work in progress. It can be obtained from Github at github.com/typst/typst.

I am a long time user of LaTeX, in particular with LyX and while not a programmer but rather an obstetrician/gynecologist, I'm computer-literate enough to generate and use templates with perl and bash. This will be an introductory presentation, showing the comparison of some simple texts in LaTeX and typst.

Frank Mittelbach

The LaTeX Companion, 3rd edition — Anecdotes and lessons learned

During the last five years a lot of work has gone into producing a new edition of *The LaTeX Companion*. In this talk I will talk about some aspects of that work, the unique challenges and some of the lessons learned during that endeavour.

Frank Mittelbach

38 years with LaTeX — A personal picture story

As the title indicates, this is part of the story of LaTeX in pictures, as seen from my eyes. It shows many highlights throughout the years and puts faces to names — some of which are in the audience but many not. It is based on what was available in my photo archive and certainly biased, but I nevertheless hope it is of some interest.

Vít Novotný

Markdown 3: What's new, what's next?

Plain TeX, expl3, and Lua provide a common programming environment across different TeX formats. Similarly, the Markdown package for TeX has provided an extensible and format-agnostic markup language for the past seven years. In this talk, I will present the third major release of the Markdown package and the changes it brings compared to version 2.10.0, which I presented at TUG 2021.

In my talk, I will target the three major stakeholders of the Markdown package:

1. Writers will learn about the new elements, which they can type in their Markdown documents.

2. Coders will learn how they can extend Markdown with new elements and how they can style Markdown documents in different TeX formats.

doi.org/10.47397/tb/44-2/tb137abstracts

3. Developers will learn about the implementation details of the Markdown package and will have a chance to discuss plans for the future governance and development of the Markdown package.

samcarter

The `tcolorbox` inner beamer theme

The `tcolorbox` inner beamer theme is a new theme for the beamer class. It replaces normal beamer blocks with `tcolorbox`es of the same look and feel. This allows users to easily modify the appearance of blocks. In this short talk, I will give a short overview of the theme and show some examples of how one can customise blocks.

Jan Šustek

On generating documented source code by blocks in TEX

In this talk I will focus on literate programming in TEX — writing source code and its documentation in a single file. Firstly I will show an easy modification of OpTEX macros to allow literate programming. Then I will modify the macros to build the source code by nested blocks which can be built consecutively in the whole document — quite similar to `tex.web`, but implemented completely in TEX. Such documentation is more comprehensible to the reader.

With a few more macros or hooks, one can apply this method in the following real situations.

- Cross references make `goto` jumps easy in programming languages with line numbers.
- The abovementioned blocks can imitate subprograms with arguments in programming languages where they are not allowed.
- TEX macros can define a metalanguage and generate the source code in two different programming languages simultaneously.

Without the TEX methods the solutions would be more complicated.

Joseph Wright

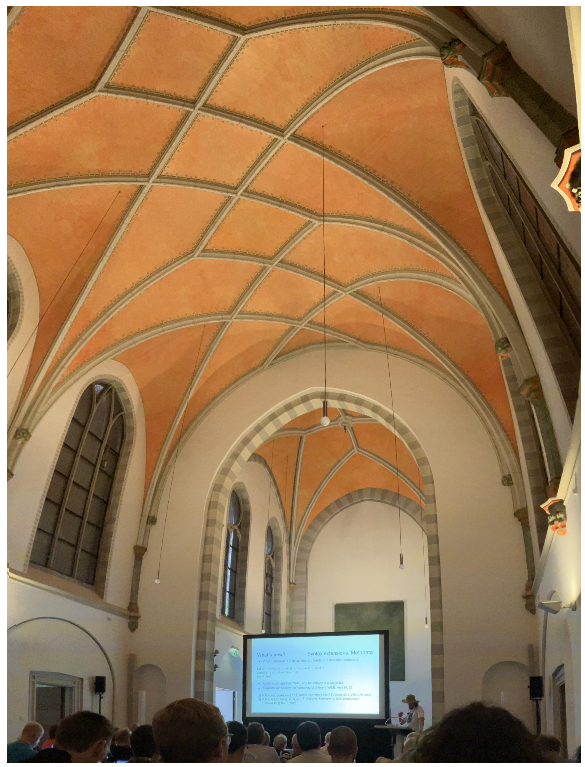
Supporting backends in expl3

The backend in TEX is responsible for the parts of producing output that TEX doesn't know about, for example colour, image inclusion and hyperlink creation. Each backend has its own syntax and range of supported concepts, so at the macro level there needs to be the appropriate code to 'talk' to the backend. In expl3, we have developed a consistent set of backend support files, based on the experience of (LA)TEX developers over 30+ years of working with these backends. Here, I will look at the history of backend abstraction and the model used in expl3.

Joseph Wright

Further adventures in Unicode-land: Refining case changing

Getting text processing right for Unicode in TEX is a challenge, particularly where one wants to support the full range in pdfTEX. Over the past few years, I have worked on one aspect: case changing. Code to carry out the Unicode case changing algorithm was integrated into the LATEX kernel a couple of years ago. Since then, we have been refining the details, adding more power and discovering new issues. Here, I'll look at what we've done to get the code working smoothly, and look forward to what might still be improved.



Lecture hall ceiling courtesy Alexander Willand.

Conference closing courtesy Reinhard Kotucha.

ArsTEXnica #34 (May 2023)

ArsTEXnica is the journal of GuIt, the Italian TEX user group (www.guitex.org).

Claudio Beccari, Editoriale [From the editor]; p. 5

Grazia Messineo, Salvatore Vassallo, Come scrivere un libro e sopravvivere [How to write a book and survive]; pp. 7–19

In 2022, we wrote with two other colleagues a book on calculus for first year economics students. Writing this book posed various problems, which can be divided in two areas: the need to use a package to collaboratively write the book, possibly with the files saved on a cloud service, with authors working at different times, but sometimes at the same time; and the need to satisfy the stylistic requests of the editor, some of which were already known and others only becoming known during the proofreading phase.

We found, for a great number of these problems, some tricks and solutions that we want to share in this paper with those willing to begin a similar job. Let us specify that these are the solutions which allowed us to survive and to produce a book satisfying all the characteristics imposed by the editor. Perhaps these are not the optimal solutions, nor unique ones.

Emmanuele F. Somma, Semplificare LATEX con ORG-mode in Emacs [Simplifying LATEX with ORG-mode in Emacs]; pp. 21–48

In writing academic or technical articles, you can reduce complexity by using markup or configurations, dropping the LATEX markup language and adopting ORG-mode in Emacs while retaining LATEX's high typographical quality. This article shows how the ORG markup exported to LATEX works. Also, it explains the attributes of the most common elements of academic papers, such as links, tables, notes, figures, and bibliographic references. Finally, in the appendix, as a practical project, the author's choices for delivering this article following the typographical rules of *ArsTEXnica* are documented.

Luigi Scarso, CJK Unicode strokes e radicals con MFLua: uno studio preliminare [CJK Unicode strokes and radicals with MFLua: A preliminary study]; pp. 49–60

We present an original method for generating an OpenType font for the Unicode block Kangxi Radicals and CJK Strokes of the Chinese writing system using MFLua and some auxiliary programs. Given its non-definitive status, the generated font is not intended to be used in normal applications.

Joseph Wright, Mapping to individual characters in `expl3`; pp. 61–62

[Published in *TUGboat* 43:3.]

Richard Koch, TeXShop, Version 5; pp. 63–66

[Published in *TUGboat* 43:3.]

Richard Koch, Interactive content using TEX4ht; pp. 67–78

[Published in *TUGboat* 43:3.]

[Received from Massimiliano Dominici.]

La Lettre GUTenberg 50, 2023

La Lettre GUTenberg is a publication of GUTenberg, the French-language TEX user group (`gutenberg-asso.org`); published online at `publications.gutenberg-asso.fr/lettre`.

Issue #50 was published June 14, 2023.

Patrick Bideault, Maxime Chupin, Éditorial [Editorial]; pp. 1–2

Patrick Bideault, Activité récente de l'association [The various works of the group]; pp. 3–5

Denis Bitouzé, Exposés mensuels sur (LA)TEX et autres logiciels [Monthly presentations on (LA)TEX and other software]; pp. 6–7

Patrick Bideault, Denis Bitouzé, Maxime Chupin, Yvon Henel, Et maintenant, une bonne *vieille* veille TEXnologique ! [TEXnology watch]; pp. 7–21

58 new CTAN packages, January–June 2023.

Maxime Chupin, Un site dédié aux exemples METAPOST [A website dedicated to METAPOST examples]; pp. 22–26

This new website is hosted at: `metapost.gutenberg-asso.fr`.

Maxime Chupin, Composer l'arbre de Huffman avec METAPOST et METAOBJ [Typesetting the Huffman tree with METAPOST and METAOBJ]; pp. 26–41

About the author's new package: **`huffman`**.

Denis Bitouzé, Dans le labyrinthe infernal des environnements d'amsmath [In the hellish maze of the amsmath environments]; pp. 42–46

A new diagram based on a flowchart by Stefan Kottwitz.

Maxime Chupin, La fonte de ce numéro : Noto [This issue's font: Noto]; pp. 47–52

Steven Matteson, En route vers Noto [The road to Noto]; pp. 53–73

Translation of the article published in *TUGboat*, Volume 41 (2020), No. 2, pages 145–154.

Patrick Bideault, Maxime Chupin, En bref [At a glance]; pp. 74–79

Short news about spams, TikZiT, exhibitions and more.

Yvon Henel, Rébus [A rebus and the solution to the rebus in the previous Lettre]; p. 80

[Received from Patrick Bideault.]

ConTEXt Group Journal 2021

The ConTEXt Group publishes proceedings of the ConTEXt meetings: `articles.contextgarden.net`.

Henning Hraban Ramm, Editorial note; p. 4

Day plan; pp. 5–6

Willi Egger, Handling fonts in ConTEXt; pp. 7–19

Obviously when typesetting electronically, we need to have at least one font available for use as base font. Although there are quite a number of fonts delivered with the distribution, often we need to use a custom font. It is not that complicated to use a third-party font; however, one needs to implement it in a structured way in order to make it available to TEX. The following article gives insight into the basic principles of how this can be done in ConTEXt.

Hans van der Meer, Macros and Lua snippets; pp. 20–32

A module containing a number of helper macros is described, many of them programmed in Lua.

Hans van der Meer, Translations from a vocabulary; pp. 33–40

This module was formerly part of `hvdm-xml` but has now been split off into an independent module with its own description. It is used for making other modules language-sensitive and is especially tailored for XML use.

Denis Maier, Automatic suppression of unwanted ligatures: The ConTEXt LMTX approach; pp. 41–47

It's well known that ligatures should be avoided in certain cases. But especially when typesetting German, ligatures have proven to be tricky beasts, and it's hard to get them right automatically.

Taco Hoekwater, MetaPost paths and pairs; pp. 48–92

This talk and paper tries to explain everything related to paths, pairs, pens and transforms in MetaPost. A fair bit of familiarity with MetaPost's data types and general syntax is assumed. In particular, I assume you have read my 'sparks, tags, suffixes and subscripts' article.

Henning Hraban Ramm, Der erweiterte Orbit [The extended orbit]; pp. 93–96

The theme of the 15th ConTEXt meeting was "expanding orbits" and took place from September 20th to 25th, 2021 at 's Sjetootje in Bassenge-Boirs (Belgium), a village between Maastricht and Liège.

Harald König, Henning Hraban Ramm, Meeting impressions; pp. 97–99

Abstracts without papers; pp. 100–102

Participant list of the 15th ConTEXt meeting; p. 103

[Received from Henning Hraban Ramm.]

Die TEXnische Komödie 2/2023

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`dante.de`).

Martin Sievers, Einladung zur 65. Mitgliederversammlung von DANTE e. V. am 13. Juli 2023 in Bonn [Invitation to the 65th user group meeting of DANTE e. V. on July 13th, 2023, in Bonn.]; p. 5

Oliver Rath, Einladung zum bayerischen TEX-Stammtisch BayTEX 2023 [Invitation to the Bavarian TEX user meeting BayTEX 2023]; pp. 6–7

Invitation to the Bavarian TEX user meeting, BayTEX 2023, taking place on July 28th and 29th in Kirchheim.

Keno Wehr, LATEX und Schulphysik 2: Schaltbilder [LATEX and physics in school 2: Electronic circuits]; pp. 8–18

How to typeset electronic circuits with several different packages.

Ralf Mispelhorn, Musik-CD mit Cover produzieren [Producing music CD covers]; pp. 19–24

A tutorial on how to design CD covers with LATEX.

Rolf Niepraschk, Mehrsprachigkeit: Das LATEX-Paket `translator` [Multiple languages with the `translator` package]; pp. 25–26

A tutorial on how to implement terms within a document to various languages.

Willi Egger, PocketDiary – Ein Modul für Terminkalender [PocketDiary — A module for diaries]; pp. 26–38

A ConTEXt module to typeset pocket diaries.

Willi Egger, Andere Kalender mit dem PocketDiary Modul [Other calendars made with the PocketDiary module]; pp. 38–56

How to design other calendars with the PocketDiary module.

Henning Hraban Ramm, ConTEXt kurz notiert! [ConTEXt news]; pp. 57–59

News regarding ConTEXt.

Sudhir Rao and Indranath Sengupta, Die Morgendämmerung der Strenge in der Kunst des Programmierens [The dawn of rigour in the art of programming]; pp. 60–90

German translation of the text published at `https://bhavana.org.in/the-dawn-of-rigour-in-the-art-of-programming`. Biographical sketch of Donald Knuth with an extensive interview.

Jürgen Fenn, Neue Pakete auf CTAN [New packages on CTAN]; pp. 56–61

List of new packages on CTAN.

[Received from Uwe Ziegenhagen.]

Zpravodaj 2023/1–2

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`.

Petr Sojka, Úvodník [Introductory word]; pp. 1–3

Go forth and participate in CSTUG to make the bright future of TeX & Friends a reality! *You can!*

Vít Starý Novotný, Nápadovník jmen pro tvůrčí psaní v LuaTeXu [Character name generator for creative writing in LuaTeX]; pp. 3–38

A famous dictum of the computer scientist Phil Karlton says that there are only two difficult things in computer science: cache invalidation and naming things. This is also true in creative writing, where authors have to come up not just with a story and a setting but also the names of all their fictional characters. In this article, we develop a language model in LuaTeX which allows authors to automatically generate names for their characters. Besides creative writing, we also discuss other uses of language models in LuaTeX, namely the automatic switching of hyphenation patterns based on the current language and blind text generation. For the TeXnically-minded users, the article acts as an introduction to the programming languages of Lua and *expl3*, and also the *xparse* LaTeX package for defining document commands in LaTeX.

Karel Šebela, Sazba hudebních skladeb [Musical composition typesetting]; pp. 39–47

TeX is a useful tool for text typesetting; however, it doesn't feature good support for music composition typesetting at its core. The TUG 2022 conference featured a talk on notation typesetting which compared the *MusiXTeX* package with the *MuseScore* and *Flat* tools, but did not mention the *PMX* and *M-Tx* preprocessors. In this article, I compare typesetting using *MusiXTeX* and its preprocessors and describe its usage. In addition, I describe the incorporation of note symbols into a paragraph text. After reading the article, the reader will be able to create a short simple excerpt of a piece of music and incorporate it into a TeX document, as well as add musical symbols to a written text.

Matúš Vančík, Pohľad TeXového nováčika na prezentáciu „Bricks and Jigsaw Pieces“ z TUGu 2022 [A TeX newbie reports on the “Bricks and Jigsaw Pieces” talk at TUG 2022]; pp. 48–53

The article deals with *TikZbricks* and *jigsaw* packages: their use, design, and possible applications. I look at these aspects of the packages through my own experimentation as a newish TeX user and I show why such packages are needed in the world of TeX.

Peter Wilson, Mělo by to fungovat XIII [It might work XIII]; pp. 54–60

Published in *TUGboat* 32:3 as “Glisterings”.

[Received from Vít Novotný.]

The gods smile at me: *The LaTeX Companion*, third edition, and ChatGPT

George Grätzer

1 Contemplation

The fifth edition of my *Math into LaTeX* book is still selling well, but it is seven years old (LIT). A lot has changed. All TeX files, BibTeX and MakeIndex files changed from ASCII to UTF-8. Lots of work has been done for references, further developing and enhancing BibTeX. And a lot more ...

So I started contemplating a sixth edition, a huge undertaking. For instance, the BibLaTeX manual alone is about 350 pages and I would have to read at least ten like that.

And then the gods smiled at me ....

First smile: *The LaTeX Companion*, third edition.

Second smile: ChatGPT.

2 The LaTeX Companion, third edition

This is a heavy contribution by Frank Mittelbach (mostly); the two volumes weigh 3.5 Kg and they run to almost 1,800 pages. I complained how much I would have to read for my sixth edition, multiply that by ten, twenty, or more for this. Frank is a voracious reader with an immense knowledge of LaTeX. The book discusses about 500 packages.

It caters to two audiences, the general user, GU (they use LaTeX for their work but cannot read TeX code, like me) and the 1% (they read TeX code and design style files and create packages).

It is hard to tell whether the 1% is 1% of LaTeX users. CTAN has currently about 6,500 packages and 3,000 contributors. We have no hard numbers on the number of research mathematicians who use LaTeX or the number of computer scientists, physicists, chemists, and so on, who also use it. So I will use 1% as a label, not quantified.

The needs of the GU and the 1% are very different. I wrote over 270 research papers, all for mathematical journals, style files supplied by the journals and 31 books, all with custom style files written by the 1%.

doi.org/10.47397/tb/44-2/tb137gratzer-smile

Now the content, it starts with Chapter 2, "The Structure of a LaTeX Document". The first few sections are for the GU, each veering off into topics for the 1%. At least one section, "Splitting the source document into several files", was made obsolete by the increasing speed of computers. When I started TeXing, it took more than two minutes to typeset a page. Now the whole sixth edition typesets in less than 3 seconds. So the section on how to use the LaTeX commands `\include` and `\includeonly` is much more esoteric.

There is little in the 130 pages of Chapter 3, "Basic Formatting Tools — Paragraph Level", for the GU. It does mention `xspace` and `microtype`, *so* important.

Chapter 4 is "Basic Formatting Tools — Larger Structures". Another 100 pages that does not have much for the GU. It does discuss modifying the list environments; not many users like how they are spaced.

Chapter 5, "The Layout of the Page", is full of useful information for the 1%.

Chapter 6, "Tabular Material", helps the reader to publish nicer-looking tables, the LaTeX default is rather ugly. It mentions the `booktabs` package I use for all my tables in my LaTeX books.

Chapter 7, "Mastering Floats", addresses every LaTeX user's huge problem: the figures and illustrations act up, they are not inserted into the document well. It suggests the use of the `\clearpage` command. In the sixth edition, I recommend trying to combine two or three floats into one.

"Graphics Generation and Manipulation" is discussed in Chapter 8, including the very important `graphicx` package of David Carlisle.

Maybe one of the most illuminating parts is Chapter 9, "Font Selection and Encodings". Everybody can benefit from reading the first five sections, a detailed introduction to the New Font Selection Scheme (NFSS) of Frank Mittelbach and Rainer Schöpf, which revolutionized LaTeX.

This completes Volume 1, and we start the second volume with Chapter 10, "Text and Symbol Fonts". It is continued in Chapter 12, "Fonts in Formulas". I am not sure for whom these chapters are meant. They must have been hard to compile, but who will benefit? The exception is the Lucida font, in which the book is typeset. Almost 200 pages of font samples seems a little excessive.

Chapter 11, "Higher Mathematics", is compulsory reading for all mathematicians. Also for all scientists using formulas. Maybe, "Typesetting Formulas" would have been a better title. College math does not utilize formulas?

Chapter 13, "Localizing Documents", is also compulsory reading for all LaTeX users who want to write in a language other than English. There is support for 30 languages, French and German are the best.

Chapter 14 discusses "Index Generation" for books and long documents utilizing MakeIndex. All book authors should read it.

Chapters 15, "Bibliography Generation" and 16, "Managing Citations" are also must-reads for book authors. The first focuses on BibTeX and the highlight of the second is the section on BibLaTeX, a 45 page summary of a 350 page user manual.

Chapter 17, "LaTeX Package Documentation Tools", Appendixes A, "LaTeX Overview for Preamble, Package, and Class Writers" and B, "Tracing and Resolving Problems" are, of course for the 1%.

This book is a *tour de force* by Frank Mittelbach, seven years of intense work. I am not sure that "the Companion would serve as the only reference needed to get 'the job' done." But for sure, no LaTeX user should be without it.

A minor complaint: The Contents is almost 20 pages long, hard to navigate. Maybe a Short Contents would have been helpful.

3 ChatGPT

For a few months now, I have been writing the sixth edition of my book, with a new title, *Text and Math into LaTeX*. Every day I consult ChatGPT dozens of times. I mostly converse about LaTeX topics but I also check English usage often (and love to write Hungarian poems in the poet Petőfy's style.)

ChatGPT got a bad rep. Version 3.5 used its pattern recognition and completion abilities to generate references and links, with unfortunate results. I use Version 4.0 which does not have this problem, nor do I ask anything where such a problem would arise.

In the first three editions of my book, I distinguish between LaTeX commands and AMS math commands. I changed that in the fourth edition, to simplify the discussion. So the phrase: "LaTeX does not provide tools to access bold lowercase Greek letters", changed its meaning from the third to the fourth edition. Is it still true?

So I ask ChatGPT. It responds: use the `\bm` command provided by the `bm` package.

I want to refresh my understanding of the command `\fontencoding`. I ask ChatGPT:

latex \fontencoding command, use and examples

The reply:

> As of my last training cut-off in September 2021, in LaTeX, the `\fontencoding` command is used to specify the font encoding scheme. Different font encodings contain different sets of characters. By using a specific encoding, you can access the set of characters that it defines.
>
> For example, the OT1 encoding is the default in LaTeX, and it does not support accented characters. If you use an accented character, then LaTeX will build it by combining the unaccented character and a separate accent character. But this can cause problems in some situations, like hyphenation.
>
> The T1 encoding, which is part of the Cork encoding scheme, supports accented characters directly. It's often recommended to use T1 for European languages, because it improves the hyphenation of words with accented characters.
>
> Here's an example:

```latex
\documentclass{article}
\usepackage[T1]{fontenc}

\begin{document}
Grüße aus Köln!
\end{document}
```

> In this example, the `\usepackage[T1]{fontenc}` line loads the T1 font encoding. The words in the document are German and include accented characters. The T1 encoding allows LaTeX to handle these characters properly.
>
> It's worth mentioning that using `\usepackage[T1]{fontenc}` changes the font encoding for the entire document. If you want to change the encoding for a specific part of the document only, you can use the `\fontencoding` command together with `\selectfont`, like this:

```latex
\documentclass{article}

\begin{document}
\fontencoding{T1}\selectfont
Grüße aus Köln!
\fontencoding{OT1}\selectfont
Back to the default encoding.
\end{document}
```

> ln this example, the T1 encoding is used only for the German text, and then the encoding is changed back to OT1 for the rest of the document. Note that `\fontencoding` doesn't change the font immediately. It only changes the encoding that LaTeX will use the next time it selects a font, which is why `\selectfont` is needed.

And if you want more related information, ask:

And how about XeLaTeX and LuaTeX

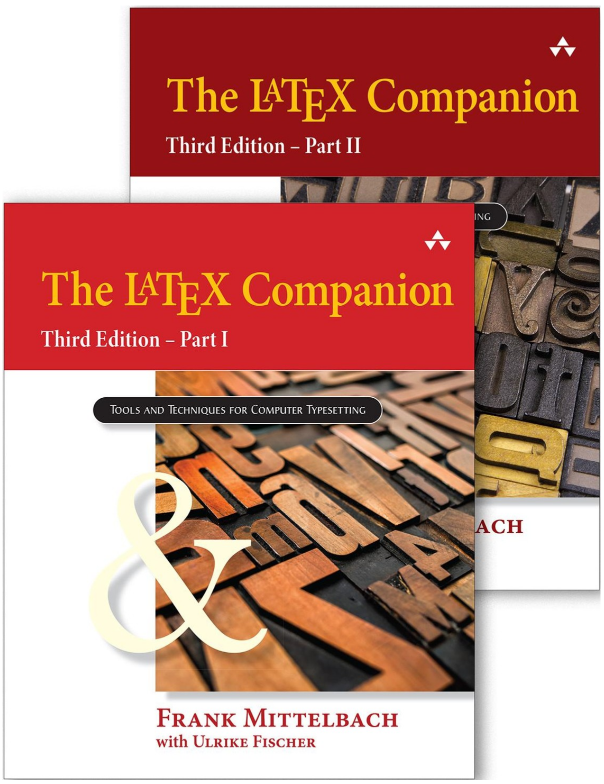
ChatGPT keeps track of all your conversations, so I have hundreds more to share. I trust these two are enough to convince you to try it for yourself.

◇ George Grätzer
University of Manitoba
gratzer (at) mac dot com
http://server.maths.umanitoba.ca/homepages/gratzer/

Book review: *The LaTeX Companion*, third edition, by Frank Mittelbach with Ulrike Fischer

John D Lamb

Frank Mittelbach with Ulrike Fischer, *The LaTeX Companion*, 3rd edition. Addison-Wesley, 2023, two volumes, 944 pp. and 970 pp., hardcover, US$96, ISBN 978-0-13-465894-0 and 978-0-201-36300-5. Electronic edition forthcoming. `tug.org/l/tlc3`



Why would you need another edition of *The LaTeX Companion* when you can find online guides to just about any available feature, e.g. [4, 5], so easily?

The LaTeX Companion, 3rd edition (TLC3), is not simply a reference manual or recipe book listing solutions to common problems you might encounter using LaTeX. That is not to say that you cannot use it as a reference manual, and many of the examples can be adapted as recipes to solve practical problems. But the real value of TLC3 is that it gives you deep insight into how LaTeX and its various packages work.

TLC3 is published in two parts, each of which is nearly as long as the second edition [3]. Part I, with contributions by Joseph Wright, introduces LaTeX and explains how to structure documents. It focuses almost exclusively on text — Part II discusses mathematics — covering such things as page layout, breaking documents into sections and paragraphs, lists, tabular material, tables of contents, footnotes and endnotes, how to include images and graphics, and floats for tables and figures. It finishes with a detailed chapter on font selection and encodings.

Part II, with contributions by Javier Bezos, Johannes Braams and Joseph Wright, starts with an illustrated guide to the fonts readily available in LaTeX. Then it discusses various packages for typesetting mathematics, followed by a chapter on how to use the various math fonts that are now possible as alternatives to the standard TeX math fonts. It also covers how to use LaTeX for languages other than English, citation and referencing, indexing, before finishing, as did the second edition [3], with a chapter on package documentation and appendices on writing preambles, packages and classes, resolving problems, and finding further resources.

There are many new developments and packages, and substantial improvements on older packages, since the second edition. The most important new developments to know about are default UTF-8 input support, LuaTeX and biber. UTF-8 is particularly important for those of us who use languages other than English. While pdfTeX is still the dominant LaTeX engine, it is worthwhile learning about LuaTeX, if only because it allows you to use OpenType font features like historical ligatures and alternative characters, shown in Figure 1. And biber is a more powerful alternative/successor to BibTeX that, especially notably, can sort correctly in languages that use characters available only in UTF-8.

Historical ligatures in OpenType

Figure 1: OpenType font features with LuaTeX.

There are too many new packages to enumerate here. So, I will point out a few that struck me as interesting. The refcheck package is a nice alternative to showkeys for showing the keys generated by `\label` commands and whether or not they are referenced. The todonotes package is also helpful for drafts: it produces *todo* notes in the margin. Figure 2 illustrates both.

A locally integrable function f is *weakly differentiable* if there exist locally integrable functions $g_1, \ldots, g_q$ from Ω to $\mathbb{R}$ such that Add definition

$$\int_\Omega f(\mathbf{x}) \frac{\partial \phi}{\partial x_j}(\mathbf{x})\,\mathrm{d}\mathbf{x} = -\int_\Omega g_j(\mathbf{x})\phi(\mathbf{x})\,\mathrm{d}\mathbf{x} \quad (1)\text{?eq:weak-diff?}$$

for $j \in \{1, \ldots, q\}$ and $\phi \in C_0^\infty$.

Figure 2: Illustration of todonotes and refcheck.

The microtype package (not new) is for those who find TeX's text formatting just not perfect enough — for example, it is what allows the hyphens on this review to jut slightly into the margin. It also can avoid rewriting to fix overfull hboxes. It only needs to be loaded to work, like subdepth, a package for better alignment of subscripts in mathematics. There is also an embrac package, for roman parentheses around italic text. The marginnote package is a flexible alternative to `\marginpar` for those who, like the authors of TLC3, realize that marginal notes are much easier to read than footnotes.[1] And the acro package lets you ensure that an acronym is defined at first point of use.

There are numerous improvements in the sciences. I will note two. The diffcoeff package simplifies typesetting of derivatives. For example, the first integral in Figure 2 can now be set with

```
\int_\Omega f(\mathbf{x})\diffp{\phi}{x_j}
(\mathbf{x})\dl{\mathbf{x}}
```

instead of

```
\int_\Omega f(\mathbf{x})
\frac{\partial\phi}{\partial x_j}
(\mathbf{x})\,\mathrm{d}{\mathbf{x}}
```

to satisfy ISO 80000-2 recommendations. The siunitx package makes it easier to type scientific units. So, we can now write `\qty{3.6}{\kilogram}` to get 3.65 kg (the weight of the two parts in hardcover) and LaTeX produces both the correct units and the correct spacing between number and units. It can also be used in a table like the following, with numbers aligned at a decimal point for easy comparison.

	median	**maximum**
mean	0.68	1.973
skewness	− 0.101	11.384
kurtosis	3.615	161.422

The `siunitx` package (or `fcolumn` if you need to match accounting conventions) lets you create such a table without resorting to fiddly `\hphantom{0}` and the like.

Chapter 8 of Part I, Graphics Generation and Manipulation, shows substantial changes from TLC2. The discussion on PostScript is much reduced, presumably because it is much less used. Instead, TLC3 now discusses some of the packages that can be used to produce graphics within LaTeX, such as those in Figure 3. This includes a reasonable introduction to tikz. The packages are outlined rather than described in the detail of [1].

[1] But, see [2] for why historians won't be switching to marginal notes any time soon.

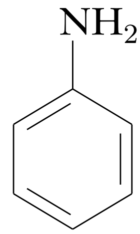


Figure 3: Graphics from packages outlined in TLC3.

One of the biggest areas of development since TLC2 has been in fonts. Chapter 10 of Part II illustrates the huge variety of fonts now available, including handwriting and symbol fonts and fonts that support languages not based on Latin. The focus is on what is either available with a standard LaTeX distribution or is freely available and good quality. But two fonts that require a license are discussed. The first is Lucida, which can be purchased from `tug.org/lucida`. The second is the Cambria font, which many LaTeX users will already have a license for. Both are suitable for both text and mathematics and support languages other than English. Lucida is a family and includes sans serif, handwriting, math (see Figure 4) and even a blackletter font. Chapter 12 discusses the many alternatives to Computer Modern Roman now available for typesetting mathematics and explains how to set them up.

The writing style, if not always elegant, is at least competent, comprehensive and candid. On the few occasions when I found myself going back through a sentence or two trying to work out what was meant, it was hard to tell if I had missed some concept because the explanation could have been better or because the concept was too subtle. The explanations are usually detailed and up-to-date — I had to update my LaTeX installation to the latest available to try out, for example, the `\DocumentMetadata` command. This command is not yet necessary for most people, but will become important as LaTeX gets better at addressing accessibility for people who cannot read printed text easily. The most important packages are explained in detail. Packages of more niche interest are outlined, with explanations on where to get further advice.

The advice in TLC3 is refreshingly honest about the limitations of LaTeX and the problems you might encounter in using various packages. It notes when there are conflicts between different packages. For example, Chapter 6 (Part I), on tabular material, discusses a range of problems in presenting tables — variations of the `tabular` environment. These include vertical and horizontal alignment of cells, issues with narrow columns, cells that span more than one row or column, automatic calculation of column widths, tables that are longer than a page, and the difficulty of rearranging the columns of a table and using tables within other environments. It then describes various ways within LaTeX to deal with these problems. It tells you which packages can be used together and which can't and presents different solutions so that you can make an educated choice. It also points out potential conflicts, for example, in using multi-page tables and floats that (currently) can only be solved manually.

TLC3 indicates potential difficulties clearly by the extensive use of marginal notes in blue. Some are marked with warning signs to indicate potential conflicts or easy misunderstandings, for example, pointing out that `[h]` does not mean 'here,' and `[bt]` and `[tb]` have the same effect when placing floats. More often, though, the marginal notes helped me find more quickly the detail I sought. TLC3 also uses a gray background when explaining features specific to XeTeX/LuaTeX or to biber so that you don't try to use what won't work with these programs and don't try to use with them what will only work in pdfTeX or BibTeX.

TeX makes it difficult to produce what is reasonably considered bad typography: random spacing, typewriter apostrophes, small changes in font size, hyphens for dashes. TLC3 explains these issues without being didactic, assuming the reader is intelligent enough to make their own judgment. For example, it includes booktabs among a range of packages for producing table rules, then suggests briefly when booktabs might be preferable.[2] It notes underlining, letterspacing and the like are not always desirable before introducing packages that produce these features and explaining what can go wrong. It explains why manual adjustment of line spacing might not work and how setspace can help. It explains why quotation can be a tricky issue, how the csquotes package can help, and later how babel allows you to follow different conventions for different languages. So, TLC3 is worthwhile reading not just to find out *how to* do something but for its discussion on whether or not what you were thinking of doing is a good idea and what the alternatives are.

One of the most convenient features of the first two editions of *The LaTeX Companion* were the tables, particularly the tables of symbols whose names you might never guess. TLC3 does not disappoint. The tables are updated. There are more of them, because there are more symbols, fonts and the like now available. There are lists of tables and, at least in the hardcover edition, each part comes with a ribbon marker you can conveniently use to mark your favorite table. There are also many figures that let you, for example, compare fonts when choosing them for your next document. And, of course, the comprehensive index nearly always leads you quickly to what you were looking for.

The LaTeX Companion, 3rd Edition, illustrates font choices such as TeX Gyre Pagella and Lucida Bright Math.

$$\int_{-\infty}^{\infty} \exp\left\{\frac{1}{2}x^2\right\} \, dx = \sqrt{2\pi}. \qquad (1)$$

Figure 4: TLC3 illustrates many font choices.

There were one or two issues I would have liked more discussion of — for example, the koma-script packages and how LaTeX might be used to produce more accessible documents. But every book must set some bounds; the first of these is well documented elsewhere, and the second remains too much work-in-progress.

In summary, then, *The LaTeX Companion, 3rd Edition* is much more than a how-to guide, recipe book or even reference for LaTeX. It is the kind of book you want to have at hand for your next LaTeX document. When you want to do something but don't know how, it will show you — and, not just show you. It will treat you as an intelligent reader. It will show you different methods and the issues around them and help you to both develop a better understanding of LaTeX and write better, more beautiful documents.

References

[1] Michael Goossens, Frank Mittelbach, Sebastian Rahtz, Denis Roegel and Herbert Voß, *The LaTeX Graphics Companion*, 2nd Edition, Addison-Wesley, 2008.

[2] Anthony Grafton, *The Footnote: A Curious History*, Faber, 1997.

[3] Frank Mittelbach and Michael Goossens, *The LaTeX Companion*, 2nd edition, Addison-Wesley, 2004.

[4] Overleaf, *Learn LaTeX in 30 minutes*. `overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes`

[5] Wikibooks, *LaTeX*. `en.wikibooks.org/wiki/LaTeX`

◇ John D Lamb
j.d.lamb (at) johndlamb dot net

[2] TLC3's own tables are mostly in booktabs style.

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 Google, US Treasury, FAO UN, Israel Journal of Mathematics, Annals of Mathematics, Res Philosophica, Philosophers' Imprint, No Starch Press, US Army Corps of Engineers, ACM, and many others.

We recently expanded our staff and operations to provide copy-editing, cleaning and troubleshooting of TEX manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

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.

▪ Design as well as LATEX implementation for e-publishing, print books and journals, or specialized projects.

▪ 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.

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`

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

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

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

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

University of Ontario, Institute of Technology, *Oshawa*, *Ontario*, *Canada,* `ontariotechu.ca`

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

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

Calendar

2023

Aug 17 – 19 TypeCon 2023, Portland, Oregon.
`typecon.com`

Aug 22 – 25 23rd ACM Symposium on Document Engineering, Limerick, Ireland.
`doceng.org/doceng2023`

Sep 8 The Updike Prize for Student Type Design, application deadline, 5:00 p.m. EST. Providence Public Library, Providence, Rhode Island.
`prov.pub/updikeprize`

Sep 10 – 16 17th International ConTEXt Meeting, Prague–Sibřina, Czech Republic.
`meeting.contextgarden.net/2023`

Oct 15 *TUGboat* **44**:3, submission deadline.

2024

Feb 4 – 7 CODEX IX, Oakland, California.
`www.codexfoundation.org`

Mar 31 *TUGboat* **45**:1, submission deadline.

Apr 16 – 20 Association Typographique Internationale, ATypI Brisbane 2024, Brisbane, Australia.
`atypi.org/conferences-events/atypi-brisbane-2024`

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, Society for the History of Authorship, Reading & Publishing, Reading, UK.
`www.sharpweb.org/main/conferences`

Status as of 15 August 2023

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568, 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`.

Reports and notices

Introductory

Intermediate

Intermediate Plus

Advanced

Reports and notices