

TUGBOAT

Volume 45, Number 2 / 2024

TUG 2024 Conference Proceedings

TUG 2024	166	Conference information and program
	168	Mitchell Gerrard / <i>TUG 2024 in Prague</i>
	172	Klaus Höppner / <i>TUG 2024 Annual General Meeting notes</i>
Accessibility	174	Simon Pfahler / <i>Easy colorblind-safe typesetting: General guidelines and a helpful L^AT_EX package</i>
	179	Changxu Duan / <i>Bridging scientific publication accessibility: L^AT_EX-markup–PDF-alignment</i>
	185	Norbert Preining / <i>T_EX (Live) and accessibility at arXiv</i>
	189	Jeffrey Kuan / <i>Legal and cultural landscape of mathematics accessibility in the United States: 2024</i>
Multilingual Document Processing	193	Boris Veytsman / <i>Extending Peter Flynn’s bookshelf package for multilanguage libraries</i>
Literate Programming	196	Mitchell Gerrard / <i>Holon programming regained</i>
Software & Tools	201	Jérémie Just / <i>On-demand production of T_EX DVDs: first feedback</i>
	203	Martin Ruckert / <i>Profiling T_EX input files</i>
	211	Vít Starý Novotný / <i>Markdown themes in practice</i>
	221	Didier Verna / <i>A large-scale format compliance checker for T_EX Font Metrics</i>
	227	Marei Peischl, Marcel Krüger, Oliver Kopp / <i>Creation of L^AT_EX documents using a cloud-based pipeline</i>
Humanities	234	Antoine Bossard / <i>A short note on typesetting Latin verse scansion with L^AT_EX and LuaL^AT_EX</i>
L^AT_EX	237	Frank Mittelbach, Ulrike Fischer / <i>L^AT_EX Tagged PDF project progress report for summer 2024</i>
	240	L ^A T _E X Project Team / <i>L^AT_EX news, issue 39, June 2024</i>
	246	Michal Hoftich / <i>Web page to PDF conversion with Rmodepdf: Leveraging LuaL^AT_EX for e-book reader-friendly documents</i>
	253	Rishikesan T, Apu V, Rajagopal CV, Radhakrishnan CV / <i>Navigating common challenges in manuscript submission: Insights for authors and publishers using Elsarticle and CAS packages</i>
Publishing	257	Jean-Michel Hufflen / <i>Making BachoT_EX proceedings—extended version</i>
	264	Andrew G. Watters / <i>Full spectrum litigator: A T_EX-themed workflow for a small litigation law firm</i>
Hyphenation	268	Ondřej Sojka / <i>Expanding hyphenation patterns across Slavic languages</i>
Fonts	271	Boris Veytsman / <i>Packaging Arsenal fonts for (X_E/Lua)L^AT_EX</i>
	274	Abdelouahad Bayar / <i>dynMath: Underlying principles of the design</i>
Abstracts	283	TUG 2024 abstracts (Fowler, Goulet, Hàn Thé Thành, Lang, Mittelbach, Obbels, Ruckert, samcarter, Snapp, Tiessen, Vaněk, Verna, Wright)
	285	<i>Die T_EXnische Komödie:</i> Contents of issue 2/2024
	286	<i>La Lettre GUTenberg:</i> Contents of issue 52 (2024)
	287	MAPS: Contents of issues 53–54 (2023–2024)
TUG Business	288	Jim Hefferon / <i>TUG bursary committee report for 2024</i>
	173	TUG institutional members
Advertisements	289	T _E X consulting and production services
	291	TUG 2024 sponsors
Cartoon News	290	John Atkinson / <i>Comics: Disaster area; Fontains</i>
	292	Calendar

T_EX Users Group

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

Individual memberships

2024 dues for individual members are as follows:

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

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

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

Journal subscriptions

TUGboat subscriptions (non-voting) are available to libraries and other organizations or individuals for whom memberships are either not appropriate or desired. Subscriptions are delivered on a calendar year basis. The subscription rate for 2024 is \$125.

Institutional memberships

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

About the cover

The cover graphic was created by Jennifer Claudio for the TUG'24 conference in Prague, in consultation with Tom Hejda, the principal organizer. The musical fragments are from Dvorák's *Humoresque*; the font, Zag Regular, is similar to that used by the venue, Hotel Grandior; the hotel facade and other Prague landmarks fill the image.

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 2024]

Printed in U.S.A.

Board of Directors

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

Arthur Rosendahl, *President*^{*}

Boris Veytsman*, *Vice President*

Karl Berry*, *Treasurer*

Jim Hefferon*, *Secretary*

Barbara Beeton*

Johannes Braams

Max Chernoff

Kaja Christiansen

Ulrike Fischer

Klaus Höppner

Tom Hejda

Jérémie Just

Frank Mittelbach

Ross Moore

Norbert Preining

Raymond Goucher (1937–2019),

Founding Executive Director

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

*member of executive committee

[†]honorary

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

Addresses

T_EX Users Group

P. O. Box 2311

Portland, OR 97208-2311

U.S.A.

Telephone

+1 503 223-9994

Fax

+1 815 301-3568

Web

tug.org

tug.org/TUGboat

Electronic mail

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

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

Volunteer T_EXnical support,
public mailing list:
support@tug.org

Contact the

Board of Directors:
board@tug.org

Copyright © 2024 T_EX Users Group.

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

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

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

2024 Conference Proceedings

TeX Users Group
Forty-fifth annual TUG conference
Prague, Czechia
July 19–21, 2024

TUGBOAT

COMMUNICATIONS OF THE TeX USERS GROUP

TUGBOAT EDITOR BARBARA BEETON

PROCEEDINGS EDITOR KARL BERRY

VOLUME 45, NUMBER 2, 2024
PORTLAND, OREGON, U.S.A.

TUG 2024 — Prague, Czechia — July 19–21, 2024

The forty-fifth annual conference of the \TeX Users Group

<https://tug.org/tug2024> ■ tug2024@tug.org

Conference committee

Tom Hejda, principal organizer

Michal Hoftich

Sophia Laakso

Ondřej Sojka

Petr Sojka

Karl Berry

Conference artwork: Jennifer Claudio

Sponsors

\TeX Users Group

CSTUG

DANTE e.V.

Google

Overleaf

Pearson Addison-Wesley

STM Document Engineering Pvt Ltd

with generous assistance from many individual contributors.

Thanks to all!



Participants

Giedrius Andreikėnas, VTeX

Nelson Beebe, University of Utah

Doris Behrendt, DANTE e.V.

Antoine Bossard, Kanagawa University

Johannes Braams, TExniek

Erik Braun, CTAN

Ben Davies, Overleaf

Changxu Duan, TU Darmstadt

Gert Fischer, Bär Backup Crew

Ulrike Fischer, LATEX Project

Stephen Fulling, Texas A&M University

Mitchell Gerrard

Vincent Goulet, Université Laval

Enrico Gregorio, Università di Verona, LATEX Project

Max Günther

Tom Hejda, Overleaf

Ahmed Hindawi

Klaus Höppner, DANTE e.V., TUG

Michal Hoftich

Jean-Michel Hufflen, FEMTO-ST

Oliver Kopp, JabRef e.V.

Marcel Krüger, LATEX Project

Jeffrey Kuan, Tailor Swift Bot

Jaromír Kuben, University of Defence

Sarah Lang, University of Graz

Carla Maggi

Frank Mittelbach, LATEX Project

Erik Nijenhuis

Norbert Preining, arXiv

Wim Obbels, KU Leuven

Marei Peischl, peiTEx, Island of \TeX

Simon Pfahler, University of Regensburg

Rajagopal C V, STM Document Engineering Pvt Ltd

Aravind Rajendran

Daniel Renschler

Rishikesan Nair T, STM Document Engineering Pvt Ltd

Martin Ruckert, Munich University of Applied Sciences

Volker RW Schaa, DANTE e.V.

Shanmugam Pillai A M, STM Document Engineering Pvt Ltd

Petr Sojka, Masaryk University

Ondřej Sojka, Masaryk University

Vít Starý Novotný

Hàn Thê Thành, Trivic s.r.o.

Tyge Tiessen

Julija Treščenko, VTeX

Christof Ullwer

Jan Vaněk, Trivic s.r.o.

Didier Verna, EPITA Research Lab

Boris Veytsman, Chan Zuckerberg Initiative,
George Mason University, TUG

Ulrik Vieth

Andrew G. Watters, Andrew G. Watters, Esq.

Alan Wetmore

Joseph Wright, LATEX Project

Jiri Zlatuska, Masaryk University

Anonymous x2

TUG 2024 program

Thursday, July 18	14:00 <i>LaTeX developers' workshop: Tagging PDF documents, Faculty of Mathematics and Physics, Sokolovská 49/8</i> 18:30 <i>Reception at Hotel Grandior (until 19:30)</i>	
Friday, July 19	08:30 Boris Veytsman, T _E X Users Group 09:00 Norbert Preining 09:30 Mitchell Gerrard 10:00 Vincent Goulet, U. Laval 10:30 <i>Break</i> 11:00 Martin Ruckert, Munich Univ. of Appl. Sci. 11:30 Tyge Tiessen 12:00 Didier Verna, EPITA Research Lab 12:30 <i>Lunch</i> 13:15 <i>Participant group photo</i> 13:30 Boris Veytsman, Chan Zuckerberg Initiative, George Mason Univ., TUG 14:00 Vít Starý Novotný 14:30 Wim Obbels, Bart Snapp, Jim Fowler; KU Leuven, Ohio State Univ. 15:00 Sarah Lang, Univ. of Graz 15:30 <i>Break</i> 16:00 Oliver Kopp, Marcel Krüger, Marei Peischl; JabRef e.V., I _A T _E X Project, peiT _E X & Island of T _E X	Welcome T _E X (Live) at arXiv Holon programming regained You (S)wave? Well, (S)tangle now! Profiling to T _E X input files Rewriting T _E X today A large scale format compliance checker for T _E X font metric files <i>Packaging Arsenal fonts for X_EI_AT_EX and LuaI_AT_EX</i> Markdown themes in practice Ximera interactive math educational resources for all: From I _A T _E X source code to PDF, HTML and beyond I _A T _E X in the Digital Humanities Tutorial: Creation of I _A T _E X documents using a cloud-based pipeline
Saturday, July 20	08:30 Michal Hoftich 09:00 Simon Pfahler, Univ. of Regensburg 09:30 Changxu Duan, TU Darmstadt 10:00 Joseph Wright, I _A T _E X Project 10:30 <i>Break</i> 11:00 Norbert Preining 11:30 Ulrike Fischer, I _A T _E X Project 12:00 Frank Mittelbach, I _A T _E X Project 12:30 <i>Lunch</i> 13:30 Jeffrey Kuan, Tailor Swift Bot 14:00 Vincent Goulet 14:30 Jean-Michel Hufflen, FEMTO-ST 15:00 Rishi T, Apu V, Rajagopal CV, Radhakrishnan CV, STM Document Engineering Pvt Ltd 15:30 <i>Break</i> 16:00 Andrew G. Watters 16:30 samcarter 17:00 Rishi T, Rajagopal CV 19:00 <i>Banquet, Restaurace Tiskárna, Jindříšská 22</i>	Web page to PDF conversion with Rmodepdf: Leveraging LuaI _A T _E X for e-book reader-friendly documents Easy colorblind-safe typesetting: General guidelines and a helpful I _A T _E X package Bridging scientific publication accessibility: I _A T _E X-markup-PDF alignment Templates: Prototype document elements arXiv's role in making research accessible Progress in the I _A T _E X tagging project: 2024 Hooks, sockets and plugs End-user usage of accessibility packages and templates A journey through the design of (yet another) journal class Making BachoT _E X proceedings Navigating common challenges in manuscript submission: Insights for authors using Elsarticle and CAS packages Full spectrum litigator The moloch beamer theme Tutorial: T _E XFolio—Manuscript preparation system using T _E X
Sunday, July 21	09:00 Ondřej Sojka, Masaryk Univ. 09:30 Antoine Bossard, Kanagawa Univ. 10:00 Jan Vaněk, Hàn Thé Thành, Trivic s.r.o. 10:30 <i>Break</i> 11:00 Boris Veytsman 11:30 Joseph Wright 12:00 <i>TUG Annual General Meeting</i> 12:30 <i>Lunch</i> 13:30 Didier Verna 14:00 Martin Ruckert 14:30 <i>Closing & final break</i> 16:00 <i>Organ concert, St. Salvator church, Salvátorská 1</i>	Expanding hyphenation patterns across Slavic languages On typesetting Latin verse scansion with I _A T _E X and LuaI _A T _E X Exploring Primo: A developer's perspective Extending Peter Flynn's bookshelf package for multilanguage libraries siunitx development continues: 2024 A couple of extensions to the Knuth-Plass algorithm The color concept of HiT _E X

TUG 2024 in Prague

Mitchell Gerrard

The entire conference was streamed at no charge on YouTube. Each day's complete stream, and eventually each talk as an individual video, can be accessed via youtube.com/@TeXUsersGroup.

Thursday, July 18

TUG 2024 was in Prague, Czechia, home of beautiful architecture, beloved authors, and many brilliant \TeX nicians. The conference began with a reception at the event's venue, the Hotel Grandior.

Friday, July 19

Boris Veytsman opened the conference, welcoming the expectant attendees; then: we were on our way.

Norbert Preining gave a prerecorded video talk about \TeX on arXiv. Operating since 1991 from Cornell, arXiv is the oldest and largest server hosting preprints of largely scientific publications, with no paywall. Norbert talked about supporting old software, including multiple versions of \TeX , and gave an overview of the new submission software, which is being rewritten in a combination of Python and Docker containers, the goal being easy and painless submission to arXiv.

Next, I rooted around through the historical precursors of literate programming. An early WEB-like paradigm called “Holon Programming” was described in a hard-to-find technical report by Pierre-Arnoul de Marneffe, but it is now typeset and freely available online.

Vincent Goulet presented on a literate programming workflow he developed for the R ecosystem, in the context of teaching courses related to actuarial science. A literate setup allows Vincent to keep teaching materials for himself, students and TAs in one file, which includes: lecture notes, questions and solutions, and unit tests. He described a circular dependency issue he ran into in this setup, and provided a clever way out. (During the talk a tiny spider appeared in front of me, suspended from the high conference ceiling, in search of weaving some web.)

Martin Ruckert described a profiler he wrote for \TeX . A profiler is a tool to tell you where software spends the majority of its time when running on some input, so a developer can know where to apply optimizations. Martin said we should never optimize for speed without a profiler. His \TeX profiler is aimed for use by macro \TeX nicians whose macros will be used frequently by many people; the document author doesn't need a profiler, unless curiosity counts as a necessity. We looked at how to optimize an input

example from the Book of Numbers, and Martin encouraged us to only apply optimizations if they can be done within a day or so.

Tyge Tiessen rewrote the entirety of $\text{\TeX}82$ in Rust (yes, it even passed the TRIP test) and he lived to tell the tale. Tyge's initial goal of learning Rust through a medium-sized project expanded a bit when he settled on reimplementing \TeX . He began with a straightforward translation of each module, using Rust's `unsafe` keyword to access globals, and creating a nice approximation of Pascal's `goto` statements. Eventually this was refactored to remove all explicit global accesses. Like Knuth's code, Tyge's version includes no external dependencies. I say: respect.

Didier Verna presented on the design of a parser within his platform for experimenting with typesetting algorithms, which is written in Common Lisp. He spoke about the need for robustness and flexibility in software design, and confessed to being a “flexibility psychopath”: there are now around a dozen recovery options in his parser. I learned of a (flexible!) error handling system in Common Lisp that improves upon traditional try/catch mechanisms by allowing you to restart computation at arbitrary program locations. Didier's parser validated nearly 80,000 font metric files, finding only two truly unusable fonts among the 770 flagged as non-compliant.

We took a break for lunch and the group photo, which included many shifts up a big set of steps as we worked to get everybody into the camera's viewfinder.

Boris Veytsman began by highlighting recent work in decolonization on reclaiming cultural traditions, including typography. Specifically—Ukrainian typography, which has a rich history distinct from Russian letterforms. Boris showed a new Ukrainian font, Arsenal, recently added to \TeX Live. The specimens shown came from Boris' personal collection of favorite Ukrainian poets and writers. Arsenal was designed to be “business-like”; Boris wondered aloud: is it OK to mix business and poetry?

Vít Starý Novotný brought us behind the scenes of producing documents for the International Software Testing Qualifications Board. This is done by populating a YAML file with easy-to-read-and-write Markdown, and then running this file through a small L^AT_EX driver to produce output. The Markdown syntax (with some extensions) has covered most use cases for ISTQB, but \TeX nicians can still jump in to this workflow and tweak away if desired.

Wim Obbels spoke about Ximera, a tool for making interactive courses and textbooks. Using one L^AT_EX source file, Ximera can generate PDF and

Mitchell Gerrard

doi.org/10.47397/tb/45-2/tb140gerrard-tug24

HTML documents. Links to Desmos can be embedded and there's nice integration with Sage. If teachers want to print out PDFs, e.g., for worksheets, a QR code can bring students to interactive explanations. Thanks to TeX-in-the-browser (is it magic? no, it's WebAssembly!), students just need a browser to start playing with Ximera's creations.

Sarah Lang talked about L^AT_EX use and non-use within the digital humanities. She outlined how this field uses L^AT_EX: for conference submissions, archaeology catalogues, and print versions of digital scholarly editions (thank goodness books haven't died out as predicted). Unfortunately many people in digital humanities see L^AT_EX as exclusionary or too computer-y, but then end up increasing complications by, e.g., hacking arcane solutions in Word. Sarah mused on ways to reduce this reticence by way of approachable tutorials and blogs.

Oliver Kopp began his talk on a cloud-based pipeline for L^AT_EX by asking: why in the world would we want such a thing? It's to avoid the shoulder-shrug-frustrations of "dunno, it worked on my machine". Oliver stepped through creating a basic package in this pipeline, and on the way featured a package dependency printer, showed how to release to CTAN using GitHub/GitLab and how to use this workflow locally with the help of Docker. The day ended with an interactive tutorial in this pipeline.

Saturday, July 20

Michal Hoftich presented on a tool that can take HTML or ePub input, remove images and ads, and output the pared-down pages to PDF. This tool's responsive design helps pages display nicely on a huge screen or on a tiny phone. The overall feel is similar to "reader mode" in Firefox. The tool can also pre and postprocess HTML, and Michal showed an example of using transformation rules to remove certain HTML elements.

Simon Pfahler opened with an effective demonstration of why disregarding colorblindness can be a problem: his title slide changed color schemes and suddenly the text disappeared! This was a simulation of colorblindness, but problems like this arise frequently, as five percent of people have some form of color vision deficiency. Simon looked at colorblind-safe design, including the most important rule: always provide information in more ways than just color. He urged developers to think about color use when designing defaults, and introduced the `colorblind` package, with color definitions of various colorblind-safe schemes.

Changxu Duan talked about the difficulties involved in getting large language models (LLMs) to

consume scientific papers. Most LLMs made for this purpose are tailored to L^AT_EX input, not PDF, which an LLM interprets as one unstructured image. Changxu helps LLMs along with reading PDFs by coming up with clever ways to convert its contents into a mix of "vanilla" L^AT_EX and Markdown, allowing LLMs to offer better summaries, related paper recommendations, etc.

Joseph Wright presented on work that's been 24 years in the making: "templates" in L^AT_EX. The idea is to give the user "instances" of a standard way to implement something, and the user can then tweak a few small parameters. There are a small number of template types (or "things"). The talk was quite interactive, with lots of calls to use some other word than "template". This caused Frank to put his head in his hands; Joseph snapped a picture of this from the podium.

Norbert Preining gave a second video talk on how arXiv makes research accessible. Online scientific work is mostly in PDF form, but if one is, e.g., a blind researcher, this file format is often unhelpful. HTML is a better solution for accessibility, with responsive design, dark mode, built-in language translations, etc. Norbert described the work done at arXiv to convert the L^AT_EX sources submitted with most uploads into HTML. There are still some issues, such as missing TikZ support, but the response from the community thus far has been overwhelmingly positive.

Ulrike Fischer presented on progress in tagging PDF documents. But as Norbert just said, HTML is generally more accessible, so why bother with PDFs? Well, not every use of PDF can be replaced by HTML. PDFs are easier to handle offline and to archive; it's a faithful representation: you can save and view it on most machines without the hassle of needing to load multiple files, cookies, session IDs, etc. Ulrike showed the status of tagging efforts in various L^AT_EX packages, and it's coming along swimmingly well. She opened with a polar bear meme and ended with a cute teddy bear slide.

Frank Mittelbach began with a simple observation: hooks are devices on which you can hang multiple things. There were no hooks in early versions of L^AT_EX, then a few were added (including the `AtBeginDocument` hook), and today there's a general hook mechanism. The new hook system reduces the brittle patching that was/is rampant in many packages. Frank then spoke about sockets, in which only *one* thing can be plugged at a time. These are useful for tightly controlled code which can either be turned "on" or "off". Didier joked about renaming sockets and hooks to "socks" and "feet".

Jeffrey Kuan brought us into the realm of U.S. laws relating to accessible documents. How can we support a typical U.S. mathematician to create documents that meet legal requirements? In just a few years, all public U.S. universities and colleges will be required to make all course material “accessible”; this may or may not include PDFs posted to arXiv. But many documents being used in courses today are still not accessible, and most instructors don’t learn about accessibility, so there needs to be a set of solutions to these problems, and quick. These could include standardized courses on L^AT_EX and on accessibility in graduate school.

Vincent Goulet was tasked by The Canadian Journal of Statistics to create a bespoke class in L^AT_EX that had a distinct look and feel from their publisher’s (Wiley) class. So Vincent revamped the old class (from 1994!), replacing crowded headers with breathable ones; stacked English/French headers with side-by-side ones; and Times New Roman/Helvetica fonts with STIX/Fira ones. Many other features make Vincent’s new class a pleasure to use and, upon T_EXing, to read.

Jean-Michel Hufflen presented on making the proceedings for BachoT_EX and *Cahiers GUTenberg*. Issue #57 of *Cahiers* was published in 2012, and after ten years, issue #58 came out. So there was a backlog of submitted articles whose compilation required all the varieties of T_EX engines. And for BachoT_EX, certain extensions were necessary, such as tables of contents in both English and Polish. Jean-Michel used a T_EX parser in Scheme to do certain programmatic tasks, and generated the article files based on a makefile. Boris interjected that “People who don’t want to use `make` just reinvent it.”

Rishi T spoke on challenges in manuscript submission from a typesetter’s perspective. His company receives thousands of articles to typeset on a monthly basis. Well-designed class files are a huge help for both authors and typesetters. But certain metadata such as affiliation fields can come in varying formats requiring a lot of manual checking by the typesetter. To streamline the process, Rishi suggested better education for document authors, automated validators/linters, and publisher participation in T_EX conferences.

Andrew Watters joined us in a video talk on using L^AT_EX in the setting of a small law firm. He didn’t want to be trapped by vendor lock-in of using, for example, Microsoft’s suite of tools; so has turned to the land of free software. We got to see inside the workflow of his firm, and with a priest-like hand gesture, we were all blessed with NDAs. Andrew stepped us through his PHP scripts that generate

L^AT_EX documents that help him create essential legal documents such as prebills for clients and pleadings for the courts.

samcarter presented on a new Beamer theme named Moloch, which is a slight variation on Metropolis, one of the most-used and iconic Beamer themes. Metropolis is lovely but now a bit rusty, being last updated on CTAN in 2017. Moloch resolves many of Metropolis’ incompatibilities, cleans up its code, and uses Beamer tooling when possible. But it’s not a 100% replacement of all the Metropolis features, hence the name change. To try Moloch, most users can simply replace `metropolis` with `moloch` in the `usetheme` command.

Rishi T and Rajagopal CV gave an interactive tutorial on T_EXFolio; unfortunately I didn’t bring my laptop so didn’t participate in this.

In the evening we all went to the banquet at Restaurace Tiskárná, some in the upper level and others of us in the comfortable cave environs. The three-course meal was superb.

Sunday, July 21

At this point in the conference I had developed an affection for the cappuccino machine in the lobby, and felt anticipatory regret that I would soon be leaving it.

Ondřej Sojka talked about expanding his previous work on hyphenation patterns in Czech to other Slavic languages. There was a brief overview of Frank Liang’s hyphenation method and the current trend toward hyphenation based on phonetics rather than on etymology. Ondřej’s goal was to improve upon subpar hyphenation patterns in Slavic languages. To do so, he used wikipedia datasets of word lists and combined phonetic hyphenation with their IPA representations.

Antoine Bossard presented on typesetting Latin verse scansion. What is scansion? It’s the identification of metrical feet within verse. “Feet”? That’s the quantity and duration of syllable groupings. Poetic analysis has a standard notation (in the form of diacritical marks) for these concepts, and Antoine showed how to implement this notation in T_EX. He demonstrated this with a passage from Virgil’s *Eneid*, which he even recited at Martin’s prompting.

Jan Vaněk and H n Th  Th nh gave a demo and explanation of their Primo tool, a WYSIWYG structural PDF editor. How does one directly edit a PDF? Well, the displayed document is actually built upon XML in the background and immediately produces PDF output in the editor. The tool is collaborative; it’s like a Google Docs but for academic publishing (whose large houses often use XML to

encode their documents). Jan dove into the details of how the collaborative updates work under the hood, using a “TriLayers” abstraction.

Boris Veysman extended the `bookshelf` package, from Peter Flynn, to handle font selection for multilanguage libraries. Boris happily declared that the original package and his extension have absolutely no practical value, but: *it's fun*. This project included hacks in `expl3` and Biber, and found ways around limitations on the number of fonts you can open at a time. Boris has plans to make the width of a book's spine vary depending on its actual size.

Joseph Wright gave a development update on the `siunitx` package. His day job is a chemist, so he works with lots of units. And he has a favorite (joule per mole kelvin). Joseph gave an overview of the long history of this package, whose latest additions include complex numbers, a new model for uncertainty, new SI prefixes, finer rounding control, and better alignment in tables. One truism for a package creator is that you can never anticipate all desires from all people, and there will be some odd requests.

Before lunch there was the TUG Annual General Meeting, whose minutes will be given elsewhere in this TUGboat issue.

Didier Verna spoke about extensions to the Knuth-Plass justification algorithm. The original algorithm uses a cost function that applies demerits based on the context of looking at two consecutive lines at a time. Didier introduces a new contextual demerit: considering the beginnings and endings of consecutive lines. The idea is to avoid consecutive occurrences of common short words such as “and”. He implemented the extension in his ETAP tool, and demonstrated its effectiveness using Grimm and Melville texts.

Martin Ruckert began by thanking samcarter for helping him update to the Moloch Beamer theme used for this presentation. Martin showed how his HINT tool allows for a bigger design space for programmatic APIs than something like PDF’s fixed format. This was demonstrated by stepping through possible color specifications within HiTEX. Colors can be nested within colors and you can specify amounts of transparency for these overlays. Martin ended festively, showing the `\TeX` logo with slightly overlapping harlequin-tinted boxes in the background of each character.

Boris brought the session to a close, thanking our team of Czech hosts: Tom Hejda, Michal Hoftich, Ondřej Sojka, and Petr Sojka; and invited us to enjoy the beautiful city of Prague.

The conference ended with an excursion to a pipe organ concert at St. Salvator church. The organist was Lukáš Vendl (with some page-turning help from conference organizer Tom Hejda), and he charmed us with the following musical incantations.

Georg Muffat, Tocatta Septima

Matthias Weckmann, Magnificat Secundi Toni

Johann Gottfried Walther, Giuseppe Torelli’s Concerto in D minor transcribed for the organ

Dieterich Buxtehude, Toccata in D minor

Johann Pachelbel, Chaconne in F minor

Johann Sebastian Bach, Prelude & Fugue in E minor

After the concert we were invited to take a look into the back of the organ where the pipes are housed, and Tom—an organist himself—gave a mini lesson on the mechanics of these marvelous machines.

Acknowledgment

I’d like to thank the TUG bursary for funding, which supported me in attending this conference.

◊ Mitchell Gerrard

`mitchell dot gerrard (at) gmail
dot com`



TUG 2024 Annual General Meeting notes

Klaus Höppner

Boris Veytsman, TUG vice president, opened the meeting in Prague, Czechia, at approximately 12:00 CEST, Sunday, July 21, 2024.

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). A few numbers have been corrected for publication here, but the results are substantively the same.

1. The TUG board of directors. (tug.org/board)
2. “Formalities”: Klaus mentioned the TUG election next year, and that executive director Robin Laakso retired in September 2023; her daughter Sophia Laakso is the new office manager. (tug.org/election)
3. “Members end of June 2024”: Klaus noted that DANTE joint memberships were resumed this year, thankfully.
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 2023.
6. “International Conferences”: Klaus reported on past and upcoming conferences, particularly noting the welcome return of BachoT_EX a couple months ago. (tug.org/meetings)
7. “T_EX Live/T_EX Collection”: Klaus reported that DVDs are no longer sent by default to members, but a volunteer group will burn them on demand. (tug.org/texcollection)
8. “Board Motions”. (tug.org/board/motions.html)
9. “Goodbye”. Klaus ended his report and resigned as TUG secretary due to other projects. We will miss him in that role and are glad he’s staying on the board.

After the slides, Boris Veytsman opened the floor to discussion of helping TUG membership, or any other topics. Tom Hejda commented that he regards the TUG web site as feeling ancient, like last century, even being difficult to find out how to become a member.

The meeting was adjourned around 12:25 CEST.

Annual General Meeting 2024 of the TeX Users Group

Klaus Höppner (secretary) for the board

July 21, 2024

Members end of June 2024

At the end of June we had 1,039 paid members, with:

- 999 renewals, 40 new (25 of them trial, 2 joint, 1 subscriber)
- -41 compared to same time last year
- 82 joint members
- 416 with electronic-only option ($\approx +33$)
- 343 with auto-renewal option
- 16 of last year’s 45 trial members renewed so far
- final numbers of last years:
 December 2023: 1,144
 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 2023

	Income	Expenses	
Membership dues	75,918	Cost of goods sold	
Product sales	3,655	TUGboat	29,540
Contributions	12,597	Software	3,120
Annual Conference	-989	Fonts	1,495
Interest	4,144	Postage	2,480
Advertising	340	Other	639
Reimbursements	-1,600	Office	
		Payroll	60,946
		Overhead	15,783
		Other	47
Sum	94,065	Sum	114,530
		Net ordinary income	-20,465

Assets and Liabilities (status end of 2023)

	Assets	Liabilities	
Checkings/Savings	168,572	Committed funds	51,538
		Member income	13,290
		Payroll	978
Sum	168,572	Sum	65,806
		Equity	102,766

Committed Funds (status end of 2023)

Fund	Amount
Bursary	2,425
CTAN	10,294
GUST e-foundry	678
L <small>A</small> T <small>E</small> X3	14,493
L <small>ua</small> T <small>E</small> X	2,192
LyX	400
MacT <small>E</small> X	4,868
PDF Accessibility	11,297
T <small>E</small> X Development owed: available:	4,891 4,000 891
Sum	51,538

International Conferences

Past

- ConTeXt meeting (Czech Rep., Sept. 10–16, 2023)
- BachoTeX 2024 (Poland, May 1–5, 2024)
- Dante 2024 (Germany, Apr. 4–6, 2024)
- Journée GUTenberg (France, Nov. 18, 2023)

Upcoming

- ConTeXt meeting (Netherlands, Aug. 17–23, 2024)
- GuIT meeting (Italy, Oct. 12, 2024)

TEX Live/TEX Collection

- TEX Live 2024 released as planned
- Team: Karl, Norbert, Siep Kroonenberg, Akira Kakuto et al.
- No more physical TEX Collection DVDs produced, but ISO images available for:
 - TEX Live
 - MiKTEX
 - CTAN snapshot
- Any of them requires a double layered blank DVD
- For those who aren't able to burn their own DVD, a group of volunteers exists to burn DVDs on demand, see <https://tug.org/dvd/>

Board Motions

- 2023.3 Discontinue TEX Collection DVD (yes: 10, no response: 5)
 2023.4 Hire Sophia Laakso as office manager (yes: 13, no response: 2)
 2023.5 Bylaws change regarding office manager (unanimously)
 2023.6 Bylaws change regarding executive committee (unanimously)
 2023.7 TUG 2024 in Prague (unanimously)
 2023.8 2024 fees (unanimously)
 2023.9 2024 budget (unanimously)
 2024.1 Ukrainian students at BachoTeX (unanimously)
 2024.2 Accepting a new institutional member (yes: 8, abstain: 5, no response: 2)
 2024.3 Site access for subscribers (yes: 11, abstain: 1, no response: 3)

TUG Institutional Members

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

tug.org/instmem

Thanks to all for their support!

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

Association for Computing Machinery, *New York, New York.* acm.org

Aware Software, *Newark, Delaware.* awaresw.com

Center for Computing Sciences, *Bowie, Maryland.*

CSTUG, *Praha, Czech Republic.* cstug.cz

CTAN. ctan.org

Duke University Press, *Durham, North Carolina.* dukeupress.edu

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

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

L3Harris, *Melbourne, Florida.* l3harris.com

LATEX Project. latex-project.org

MacTEX. tug.org/mactex

Maluhy & Co., *São Paulo, Brazil.* maluhy.com.br

Marquette University, *Milwaukee, Wisconsin.* marquette.edu

Masaryk University, Faculty of Informatics, *Brno, Czech Republic.* fi.muni.cz

Modular Font Editor K. mfek.org

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

NASA. nasa.gov

National Security Agency. nsa.gov

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

Overleaf, *London, UK.* overleaf.com

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

Tailor Swift Bot, *College Station, Texas.*

TEXFolio, *Trivandrum, India.* texfolio.org

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

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

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

Easy colorblind-safe typesetting: General guidelines and a helpful L^AT_EX package

Simon Pfahler

Abstract

Roughly 5% of people suffer from some sort of color-vision deficiency (CVD) [13]. To create documents that are accessible to anyone, it should therefore be considered how affected people perceive the colors in the documents. In colorblind-safe documents, the contents are presented in a way such that the same information is conveyed to readers regardless of potential CVDs. We first discuss how color is typically used in documents and categorize this into three different use cases that need different color schemes to convey the desired information. We then present some easy to follow rules for typesetting colorblind-safe documents. Finally, we take a look at available colors in L^AT_EX and how well they are suited for colorblind-safe documents. These considerations have lead to the development of the `colorblind` package, which we will introduce and discuss briefly.

1 Introduction

Human color perception is based on three types of cone cells in our retinas [13], which enable us to distinguish between different wavelengths of light. A variation in the sensitivity of different cone cell types can reduce or shift the color vision of affected individuals. Such differences are called *color-vision deficiencies* (CVDs), and are typically tested for using *Ishihara color test plates* [1]; see an example in fig. 1.

The most common type of CVD is *anomalous trichromacy* [13], where one cone type is less sensitive than the others, leading to aberrant color perception. The extreme case where one cone type is completely absent or dysfunctional is known as *total color blindness* [12].

People affected by a CVD typically have a harder time picking up on information conveyed through color. For example, the most common types of color blindness, deuteranopia and protanopia, make it difficult to distinguish red and green [12].

In this paper we will first discuss how color can be used in documents to encode information. We will discuss general techniques and guidelines to make documents more accessible to people with CVDs. Additionally, we will investigate how well (or rather how badly) the standard colors in L^AT_EX can be distinguished by people with CVDs. Finally, we will give some remarks on the current state of the `colorblind` package, and goals for its future development.

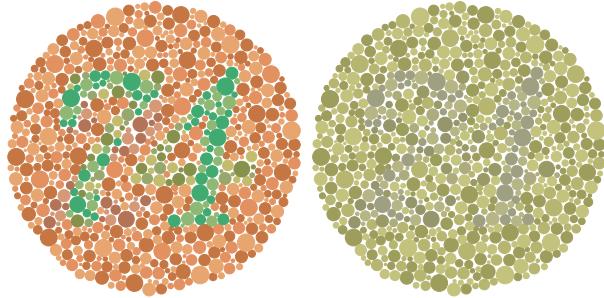


Figure 1: Ishihara test plate number 9. Left side (reads “74”) shows normal version, right side (reads “21”) shows a simulation of protanopia, a type of red-green color blindness.

2 Colorblind-safe design

When typesetting documents, we should pay attention to which color combinations we use within one visual unit. A visual unit may be a graphic, a table or a paragraph of text. The colors used in a visual unit are called the *color scheme*. In this section, we will first learn about different types of color schemes. After that, we have the necessary tools to formulate some guidelines that we can follow to achieve a colorblind-safe design.

2.1 Types of color schemes

In order to understand how to choose a suitable color scheme, we first have to understand what types of color schemes exist, and when each should be used. Let us consider different cases in which we might want to use colors to convey information. For this, we follow five fictitious students through their 20-week-long semester. At the end of each week they have to take a test which is graded from “1” (good) to “6” (bad). The average of these grades is their overall grade at the end of the semester.

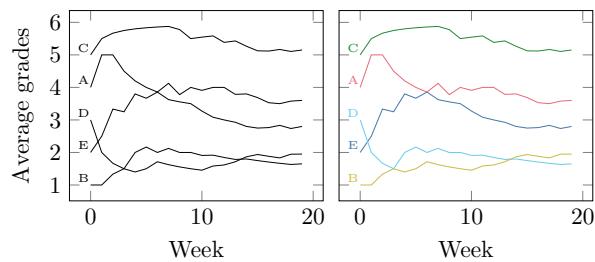


Figure 2: Example graphic showing a typical use case of a qualitative color scheme. Both graphics show the same data, but the right one uses colors to make the lines more distinguishable.

First, we might be interested in how the average grade of each of our students changes during the semester. This is plotted in fig. 2. The left graphic

does not use color, and it is hard to distinguish the lines, especially at crossings where it is unclear which line goes where. In the right graphic, the lines are colored, which makes it easier to extract the information from the graphic. Now we can ask ourselves: What does the color scheme need to satisfy in order to be helpful? The only reason we introduce color here is to help with distinguishing the lines. Such a scheme is called a *qualitative color scheme* [3], and its only goal is to provide colors that are easily distinguishable, regardless of potential CVDs.

Next, we might be interested in the individual grades of our students. They are given in fig. 3.

A	4	6	5	3	3	3	3	6	1	6	3	5	4	1	4	2	1	3	5	4
B	1	1	2	2	1	2	3	1	1	1	1	3	2	3	4	3	1	1	4	2
C	5	6	6	6	6	6	6	5	3	6	6	3	6	3	3	5	6	4	6	
D	3	1	1	1	4	3	1	3	1	2	1	2	1	1	2	1	1	1	1	2
E	2	3	5	3	6	3	5	2	3	3	1	1	2	2	1	2	3	4	1	4
Week																				
A	4	6	5	3	3	3	3	6	1	6	3	5	4	1	4	2	1	3	5	4
B	1	1	2	2	1	2	3	1	1	1	1	3	2	3	4	3	1	1	4	2
C	5	6	6	6	6	6	6	5	3	6	6	3	6	3	3	5	6	4	6	
D	3	1	1	1	4	3	1	3	1	2	1	2	1	1	2	1	1	1	1	2
E	2	3	5	3	6	3	5	2	3	3	1	1	2	2	1	2	3	4	1	4
Week																				

Figure 3: Example graphic showing a typical use case of a diverging color scheme. Both graphics show the same data, but the lower one uses colors to make the graphic easier to understand.

In the uncolored graphic, it is difficult to see differences between the students because the grade information is provided only as text. By adding color to this graphic, it becomes easier to interpret since differences between students can be observed without looking at the precise values. Again we ask ourselves, what does a color scheme need to satisfy in order to be suitable for this graphic? This time, the colors should provide a continuous range between two easily distinguishable extremes (in our case blue for good and red for bad grades). The middle color of the scheme (in our case nearly white) should be a neutral color. We call such schemes *diverging color schemes* [3], and their goal is to visualize a range of numbers [min, max] where the mid-point is considered “neutral”.

For the third type of color scheme, we consider the weekly number of questions our students ask in class, shown in fig. 4. We add color to this graphic for the same reason as in the previous case of fig. 3 — it helps us to discern patterns in the data. Even though the graphic looks almost identical to the previous one, our choice of color scheme should be different due to a subtle difference in the type of information

A	2	1	3	3	2	0	2	3	1	1	4	0	3	0	0	1	2	2	1	1
B	2	5	6	9	6	6	5	5	7	3	3	4	3	4	3	6	6	4	5	2
C	6	6	13	6	9	3	1	4	6	3	5	6	5	1	4	6	1	6	6	6
D	6	8	7	4	12	6	5	8	7	6	6	4	8	8	4	15	10	6	3	6
E	11	6	10	4	8	6	5	5	5	7	7	5	6	7	7	4	8	7	6	8

Week

A	2	1	3	3	2	0	2	3	1	1	4	0	3	0	0	1	2	2	1	1
B	2	5	6	9	6	6	5	5	7	3	3	4	3	4	3	6	6	4	5	2
C	6	6	13	6	9	3	1	4	6	3	5	6	5	1	4	6	1	6	6	6
D	6	8	7	4	12	6	5	8	7	6	6	4	8	8	4	15	10	6	3	6
E	11	6	10	4	8	6	5	5	5	7	7	5	6	7	7	4	8	7	6	8

Week

Figure 4: Example graphic showing a typical use case of a sequential color scheme. Both graphics show the same data, but the lower one uses colors to make the graphic easier to understand.

presented. Similar to before, the colors visualize numbers in a range [min, max], but this time, the mid-point is not considered neutral. That is why we use a *sequential color scheme* [3], which helps to visualize a range of numbers [min, max] where one end is considered “neutral” (this is often 0), whereas the other is considered “extreme”. Importantly, the mid-point of such schemes is not special and often arbitrary (in our example, it would change if the maximum number of questions asked by any student is higher).

2.2 Colorblind-safe color schemes

These three types of color schemes provide the basis for how we use color in documents. In order to write colorblind-safe documents, it is important that the schemes we use provide easily distinguishable colors that retain their meaning under potential CVDs. Various such color schemes exist in the literature [2, 8, 14]. As an example, we focus on the schemes designed by Paul Tol [14], as these make up the most comprehensive set of such schemes that we were able to find. As examples, fig. 5 shows the color schemes used in the example plots from section 2.1. These color schemes typically consist of the main scheme, plus an additional color that can be used for missing data.

2.3 Guidelines

CVDs appear in many different variations and grades of severity, up to monochromacy, where different colors can only be distinguished via their perceived brightness. This means that while the color schemes presented in this paper are easier to distinguish under the most common CVDs, information encoded only in color can never be completely colorblind safe. This leads us to the most important rule in colorblind-safe design [5]:

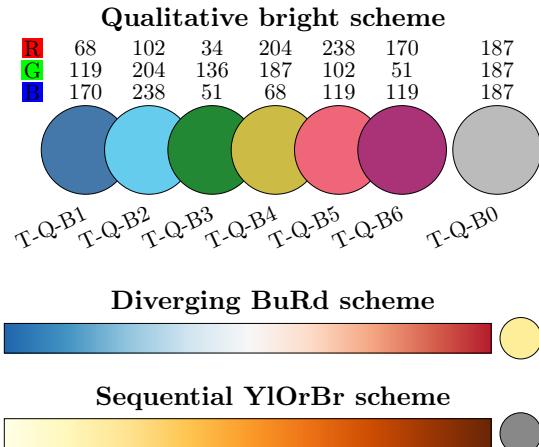


Figure 5: Example color schemes by Tol [14]. For the *qualitative bright* scheme, RGB values are given above the colors, and the color name when using the `colorblind` package is given below them.

Rule 1: Always provide information in more ways than just color.

But how does that rule work in practice? For this, let us revisit the three graphics from section 2.1. In figs. 3 and 4, the information encoded in color is the same as the numbers inside the boxes, so even if the color information cannot be picked up by some individuals, the information is still present as numbers, albeit in a more inconvenient way. These two graphics can thus be considered colorblind-safe. For fig. 2 however, without the color information, it is difficult to distinguish the lines, so we need another way to help the reader make this distinction. To achieve this, we can for example introduce different patterns in the lines, see fig. 6.

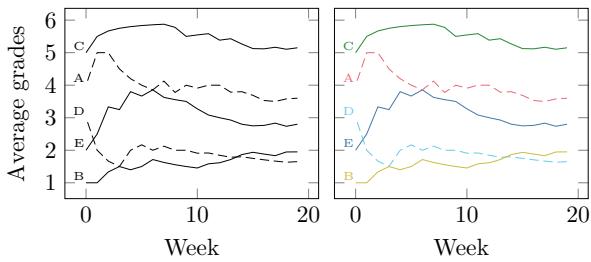


Figure 6: Improved version of fig. 2, where every second line is dashed to make lines distinguishable even when the colors are removed.

If this rule is satisfied in a document, it is by construction guaranteed to be colorblind-safe. However, this does not mean that it is *convenient* for people with CVDs to extract the information, as in the above examples from figs. 3 and 4. In order to

achieve the best possible result, a few more rules should be considered when using color.

Stick to a color scheme.

Rule 2:

- (a) Do not mix colors within a scheme.
- (b) Do not use shades of colors.

Colors within colorblind-safe color schemes are designed to be easily distinguishable for people with the most common CVDs, so we should use only colors from one color scheme in any given visual unit. By extension, even colors from the same scheme should not be mixed, since this makes it harder to distinguish them. Even if the result of the mixing is easily distinguishable for people with normal color vision, the same might not be true under certain CVDs. For the same reason, shades of colors (i.e., mixings with black or white) should be avoided, because the brightness of colors is also used to make sure the colors are distinguishable.

Rule 3: Do not use color for information and aesthetics simultaneously.

Color is often also used for aesthetic reasons, e.g., on a scientific poster. This is usually unproblematic, as the color does not convey information in this case. However, if color is used to convey information in a visual unit, avoid using additional color for aesthetic purposes, as this makes it more difficult to extract the information encoded in the color.

Rule 4: Do not use rainbow color schemes.

Due to the many different colors in a rainbow color scheme, they are inevitably difficult to distinguish under CVDs. Therefore, it is best to avoid them. If a rainbow color scheme has to be used at all cost, Paul Tol (and thus also the `colorblind` package) provides both a discrete as well as a continuous version [14], which are optimized to be as distinguishable as possible.

By following these four simple rules, we can ensure that the information encoded in a document is presented in a colorblind-safe way, and that it is reasonably convenient for people affected by CVDs to extract the information. As a side note, following these rules also leads to documents that do not suffer from information loss when printed in black and white, which is usually also desirable.

3 Colors in L^AT_EX

In this section, we take a look at how we use colors in L^AT_EX. For this purpose, we first consider the built-in colors of the standard color package `xcolor` [9], and test how well they can be distinguished under CVDs. After that, we propose the use of the new package

`colorblind`, which provides many colorblind-safe schemes of all types mentioned in section 2.1.

3.1 The standard L^AT_EX colors

When using colors in L^AT_EX, the most convenient way is to use the built-in named colors, like `red` or `green`. This is also what L^AT_EX packages like TikZ do [7]. Figure 7 shows the 19 built-in colors.

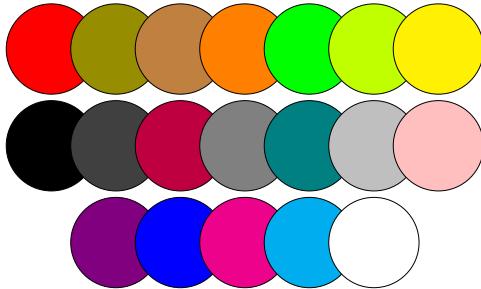


Figure 7: The 19 built-in colors from the `xcolor` package.

These colors are certainly not all easily distinguishable even under normal vision, but we can easily find over ten colors that look like they might work well together.

Now let us consider in fig. 8 how these colors look to a person affected by deuteranopia, a type of red-green blindness. The colors get collapsed into three categories, and the differences within each category are mostly down to different brightness levels. From these colors, there are maybe six distinguishable colors left that we could reasonably use in our graphics, already including black and white. Also we should keep in mind that we have considered only deuteranopia so far; considering other common types of color blindness as well would reduce the number of distinguishable colors even further.

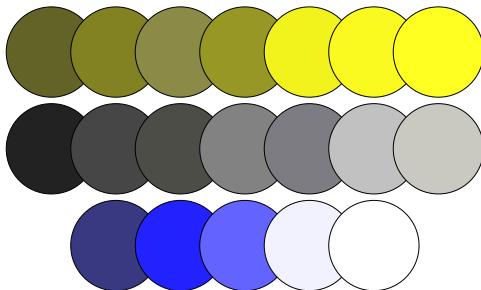


Figure 8: The 19 built-in colors from the `xcolor` package as perceived under deuteranopia, in the same order as seen in fig. 7.

Essentially, we see that the built-in colors by `xcolor` are unsuitable for use in colorblind-safe documents.

We therefore need an alternative for choosing our colors.

3.2 The colorblind package

There are many ways of obtaining color schemes that are better suited for colorblind-safe documents than the standard L^AT_EX colors. The most influential of these tools is probably *ColorBrewer* [2]. Such tools make it easy to create color schemes, but to use them in L^AT_EX, we usually have to define the colors by hand. This is cumbersome, and is probably the main reason why people stick to the standard colors instead of better alternatives.

Thus, we have introduced the L^AT_EX package `colorblind` [11], which defines color schemes that can be used in colorblind-safe documents. Figure 5 shows examples of the schemes provided by this package. The color names all start with the scheme name, e.g., T-Q-B for Tol's Qualitative Bright scheme. The scheme name is then followed by the number of the color, e.g., T-Q-B0 to T-Q-B6. In each scheme, color number 0 is associated with bad/missing data and should be used accordingly.

It might seem inconvenient at first that the colors do not have natural names like `red` or `green`, but there is a simple reason for this. Certain colors (`green`, `red`) are often used by people with full color vision to convey certain meanings (`good`, `bad`). This meaning is difficult for people with CVDs to pick up. By not using natural color names, it is easier to write colorblind-safe documents that do not make use of said connotations. Additionally, natural color names can be cumbersome, e.g., when slight variations of a color are used. It is annoying having to look up if a color is called `blue` or `cyan`.

In addition to these simple color definitions, the `colorblind` package also provides continuous pgf colormaps for color schemes that allow interpolation of their colors. These can be activated using the abbreviated form of their color scheme name, e.g., `/pgfplots/colormap name=T-D-BR` for Tol's Diverging Blue-Red color scheme.

4 Future plans for the colorblind package

As a last point in this paper, I would like to explain my idea behind the `colorblind` package and discuss how to realize this goal.

The vision I had in mind when starting work on the `colorblind` package was to create a L^AT_EX package that makes it possible to view elements of a document as they are seen by people with CVDs. This would make it easy to check if a document is colorblind-safe directly during the writing process, as opposed to current CVD simulators [4, 15], which

are available only as a post-processing step. By having a CVD visualization enabled during the writing process, common mistakes such as the use of color connotations mentioned above can easily be avoided, leading to documents where colorblind-safeness is not an afterthought, but is achieved naturally.

Such a functionality would be similar to how the `xcolor` command `\selectcolormodel{gray}` converts all colors to grayscale [9]. In fact, the most promising way to implement this feature appears to be the implementation of a new color model for each type of CVD that should be supported. As the `xcolor` package is slowly being replaced by the new `l3color` package within the new L^AT_EX kernel [10], an implementation of these new color models that builds on `l3color` is advisable, as it is more future-proof. A different approach was used for creating the CVD versions of figs. 1 and 8, where the `\color` command was redefined to convert the colors to a representation of a CVD. Unfortunately, this approach suffers from various limitations and is therefore not viable for regular use.

In addition to this idea of providing CVD simulation directly within L^AT_EX, future additions to the `colorblind` package will include options to change the default colors of commonly used L^AT_EX packages to colorblind-safe alternatives, e.g., for `pgfplots` [6].

5 Conclusion

In this paper, we discussed how color-vision deficiencies affect the accessibility of documents. Through some examples, we learned how color can be used in documents to convey information. Then, we provided some rules that help with achieving colorblind-safe documents. Finally, we discussed which colors are typically used in L^AT_EX and how we can hopefully improve the status quo by providing an easy way to use colorblind-safe colors. The current and planned features of the `colorblind` package can probably be extended by other useful features for colorblind-safe typesetting. If you have any suggestions for this, feel free to contact me.

References

- [1] J. Birch. Efficiency of the Ishihara test for identifying red-green colour deficiency. *Ophthalmic and Physiological Optics*, 17(5):403–408, Sept. 1997.
- [2] C. Brewer. ColorBrewer, 2021. github.com/axismaps/colorbrewer
- [3] C.A. Brewer, G.W. Hatchard, M.A. Harrower. ColorBrewer in Print: A Catalog of Color Schemes for Maps. *Cartography and Geographic Information Science*, 30(1):5–32, Jan. 2003. doi.org/10.1559/152304003100010929
- [4] N. Burrus. DaltonLens, 2021. daltonlens.org/colorblindness-simulator
- [5] A. Campbell, C. Adams, et al. Web content accessibility guidelines (WCAG) 2.2, 2023. www.w3.org/TR/WCAG22/
- [6] C. Feuersänger. *The pgfplots package*. Create normal/logarithmic plots in two and three dimensions. pgfplots.sourceforge.net/
- [7] C. Feuersänger, H. Menke, et al. *The pgf package*. Create PostScript and PDF graphics in T_EX. ctan.org/pkg/pgf
- [8] Y.G. Ichihara, M. Okabe, et al. Color universal design: the selection of four easily distinguishable colors for all color vision types. In *Color Imaging XIII: Processing, Hardcopy, and Applications*, R. Eschbach, G.G. Marcu, S. Tominaga, eds., vol. 6807, p. 68070O. International Society for Optics and Photonics, SPIE, 2008. doi.org/10.1117/12.765420
- [9] U. Kern, L^AT_EX Project Team. *The xcolor package*. Driver-independent color extensions for L^AT_EX and pdfL^AT_EX. github.com/latex3/xcolor
- [10] L^AT_EX Project Team. *The l3kernel package*. L^AT_EX3 programming conventions. ctan.org/pkg/l3kernel
- [11] S. Pfahler. *The colorblind package*. Easy colorblind-safe typesetting. github.com/simon-pfahler/colorblind
- [12] L.T. Sharpe, A. Stockman, et al. Opsin genes, cone photopigments, color vision, and color blindness. *Color vision: From genes to perception*, 351:3–52, 1999. www.allpsych.uni-giessen.de/karl/colbook/sharpe.pdf
- [13] M.P. Simunovic. Colour vision deficiency. *Eye* 24(5):747–755, May 2010. doi.org/10.1038/eye.2009.251
- [14] P. Tol. Paul Tol’s Notes: Colour schemes and templates, 2021. personal.sron.nl/~pault/
- [15] M. Wickline. Coblis—color blindness simulator, 2001. www.color-blindness.com/coblis-color-blindness-simulator/
 - ◊ Simon Pfahler
[simon.pfahler\(at\)ur.dtu.de](mailto:simon.pfahler(at)ur.dtu.de)
<https://simon-pfahler.github.io>
ORCID 0009-0001-7364-4005

Bridging scientific publication accessibility: \LaTeX –markup–PDF–alignment

Changxu Duan

Abstract

This paper introduces a method to enhance the accessibility of scientific publications across multiple formats. Prior initiatives have predominantly centered on transforming \LaTeX source code or PDF documents into markup languages like XML and Markdown. However, such methods typically overlook preserving the visual layout inherent in PDF pages. The approach in this paper exploits the color properties in \LaTeX code to maintain consistency and alignment between the visual presentation of documents in PDF format and the digital presentation of markup languages and \LaTeX code. This strategy not only fills in the gaps of previous approaches but also promotes the integration and accessibility of scientific document formats.

1 Introduction

Accessibility in PDFs ensures that documents are usable by everyone, including those with disabilities, by making them navigable and readable through assistive technologies. This involves structuring PDFs with metadata tags to define reading order and document elements, embedding text attributes for readability, providing alternative text for visual content, and including navigational aids like bookmarks. These measures, aligned with standards like PDF/UA, ensure that the PDFs are not only accessible but also comply with legal requirements for inclusivity [8].

Accessible PDFs benefit not only individuals with disabilities but also the broader user base. Features that make documents accessible, such as clear navigation and structured headings, improve the overall user experience and enhance the document's usability for everyone. These features make it easier to navigate through the text, find information quickly, and convert the document into other formats as needed. Additionally, the structure required for accessibility, such as tagged PDFs, aids in the correct reflow of text and associated graphics when adjusting the size of a document or its viewing mode. This adaptability is essential as digital content is increasingly accessed on a diverse array of devices, including smartphones and tablets.

\LaTeX -sourced PDFs often lack accessibility primarily due to the inherent complexity and customization capabilities of \LaTeX . The \LaTeX system allows for a vast array of macros and packages, enabling highly complex document structures that do not au-

tomatically support accessible features necessary for assistive technologies, such as structured headings and alternative text for images. Unlike modern document creation tools that include built-in accessibility features, traditional \LaTeX compilers like $\text{pdf}\text{\TeX}$ and $\text{Xe}\text{\TeX}$ do not inherently support tagging.

Moreover, even with \LaTeX packages designed to facilitate tagging, such as tagpdf [5, 15, 16], integrating these features requires significant technical expertise and meticulous configuration. The numerous \LaTeX macros can interact unpredictably, potentially undermining the structural integrity needed for accessible documents. Additionally, the \TeX engines that process \LaTeX are primarily focused on print quality, not digital accessibility. This focus, combined with a general lack of awareness among \LaTeX users about accessibility standards, further complicates the production of accessible PDFs. The responsibility for ensuring accessibility often falls on the authors, who must navigate the steep learning curve of both \LaTeX and accessibility requirements.

Other studies on \LaTeX accessibility have converted \LaTeX directly into a markup language, skipping the generation of PDF files [2]. While converting \LaTeX to markup languages directly might improve accessibility by leveraging the inherent structural and semantic capabilities of HTML or XML, it also means losing out on the robust, cross-platform fidelity and rich feature set that PDFs offer.

In this paper, I explore a method that adds accessibility to existing PDF files without manually altering the existing \LaTeX code.

2 Related work

2.1 \LaTeX to markup transformation

Currently, two tools capable of converting \LaTeX source code to markup languages are $\text{\LaTeX} \text{XML}$ [13] and Pandoc [12]. However, both tools directly convert from \LaTeX to markup languages like HTML or Markdown without compiling to PDF first. This approach bypasses some of the benefits that come from generating and utilizing PDF files, including the precise control over layout and typography that PDFs offer.

2.2 \LaTeX –PDF alignment methods

The current methods for aligning PDF and \LaTeX content typically rely on kernel patching or color-based alignment techniques. One prominent kernel-based alignment method is Sync \TeX [9], widely implemented across various \LaTeX editors. This method provides a dynamic link between the source \LaTeX code and the generated PDF, facilitating easy navigation between them. Color-based alignment methods

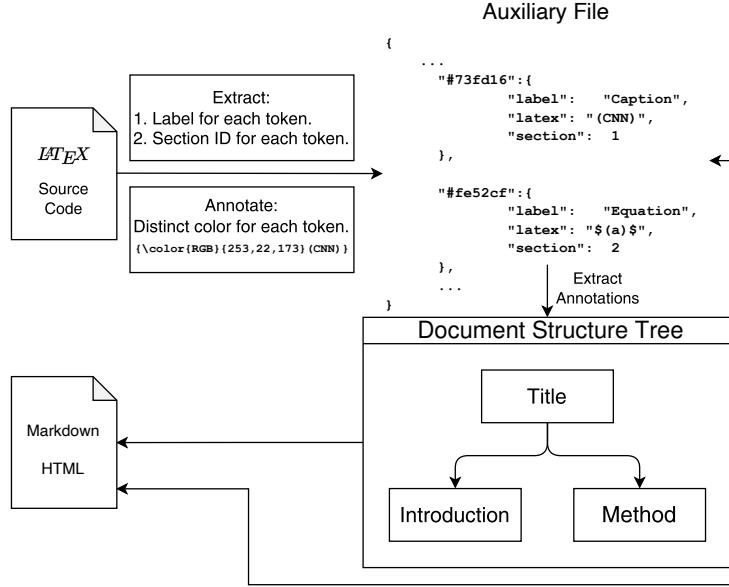


Figure 1: The overall process of accessibility annotation. The alignment of LATEX–markup–PDF is built up with auxiliary files.

include approaches like those demonstrated by the DocBank [11] dataset. In this dataset, different types of page elements are assigned unique colors, which are then used to facilitate the extraction and analysis of document layouts.

My approach builds upon the color-based alignment strategy used by DocBank. By assigning distinct colors to different elements within the LATEX code and then analyzing these in the generated PDF, I can accurately align and categorize content, enhancing the effectiveness of document layout analysis. This extension of the DocBank method allows for more detailed and precise handling of the alignment between the LATEX source and the PDF output.

3 Methodology

The method discussed in this paper is extended from LATEX Rainbow [4], a tool designed for generating Document Layout Analysis datasets. LATEX Rainbow begins by downloading a paper’s source code from arXiv, which it then processes using a Python-based LATEX parser. Each element within the body of the paper’s source code is assigned a unique RGB color using commands from the `xcolor` package. The tool maintains a dictionary to map these colors to specific element labels including title, author, abstract, math, table, body, caption, figure, and reference. After coloring, LATEX Rainbow compiles the annotated LATEX code into a PDF using a containerized compilation environment. This step ensures that the environment is consistent and reproducible, minimizing the

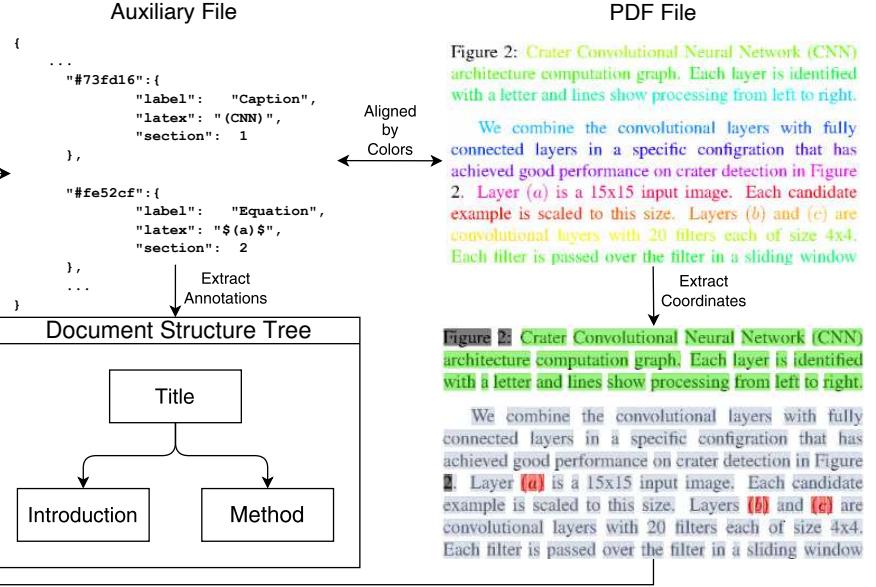


Figure 2: Crater Convolutional Neural Network (CNN) architecture computation graph. Each layer is identified with a letter and lines show processing from left to right.

We combine the convolutional layers with fully connected layers in a specific configuration that has achieved good performance on crater detection in Figure 2. Layer (a) is a 15x15 input image. Each candidate example is scaled to this size. Layers (b) and (c) are convolutional layers with 20 filters each of size 4x4. Each filter is passed over the filter in a sliding window

Figure 2: Crater Convolutional Neural Network (CNN) architecture computation graph. Each layer is identified with a letter and lines show processing from left to right.

We combine the convolutional layers with fully connected layers in a specific configuration that has achieved good performance on crater detection in Figure 2. Layer (a) is a 15x15 input image. Each candidate example is scaled to this size. Layers (b) and (c) are convolutional layers with 20 filters each of size 4x4. Each filter is passed over the filter in a sliding window

effects of system-specific variations on the compilation process. The final output is a color-rich PDF file, from which I can extract the color of each element. By extracting the color of each element in the PDF, I can align the label of each element in the PDF.

LATEX Rainbow provides processing from LATEX code annotations to PDF annotations. However, at the moment it is not directly convertible to Markup languages, for this reason, I have extended its functionality in three steps to generate accessible annotations. Figure 1 shows my annotation process.

3.1 Preprocessing LATEX code

While arXiv ensures that the LATEX source code it hosts is of high quality and guaranteed to compile, this does not necessarily mean the source is error-free. The LATEX compilation process is robust enough to handle certain syntactical inaccuracies—like unclosed curly brackets—without halting the generation of a visually correct PDF. This tolerance can mask underlying issues in the source code, such as unclosed environments or improperly ordered `\end{document}` commands, which may not disrupt the PDF output but do indicate imperfect coding practices.

These minor discrepancies, though often overlooked by the TeX engine, present challenges for the more rigid Python-based `pylatexenc` parser, which requires stricter syntactic adherence. To ensure compatibility and facilitate smoother processing with such tools, it is necessary to first identify and rectify these errors. My approach involves applying a set

of rules to filter out these syntactical errors before they reach the parser, thus enhancing the reliability of tools that are less tolerant of the flexible parsing inherent to L^AT_EX compilers.

When dealing with L^AT_EX sources, custom commands often present significant parsing challenges, especially for `pylatexenc` that requires more straightforward syntactic structures. In academic papers, it is not uncommon for authors to simplify their L^AT_EX coding by defining macros that encapsulate common L^AT_EX environments. For instance,

```
\newcommand{\beq}{\begin{equation}}
\newcommand{\eeq}{\end{equation}}
```

replace standard L^AT_EX environment tags to streamline the writing process. While these macros are fully compatible with the T_EX engine, they can complicate the parsing process for software that does not inherently interpret these user-defined shortcuts.

To address this, I use `de-macro` [7], which expands these custom commands back into their standard L^AT_EX forms. This tool not only handles the expansion of simple custom commands but also aids in consolidating L^AT_EX content spread across multiple files brought together with the `\input` command.

This preprocessing step ensures that the L^AT_EX code is transformed into a format that `pylatexenc` can accurately interpret, thereby maintaining the integrity of the document structure and content in environments that are less tolerant of such L^AT_EX customization.

3.2 Compiling L^AT_EX and extracting annotations

L^AT_EX Rainbow assigns a specific color to each element within the abstract code tree, compiles these elements into a new L^AT_EX source file, and then creates a color-coded PDF from this file. I have enhanced the method for extracting these annotations by employing two Python-based PDF parsers: `pdfplumber` [17] and `PyMuPDF`.¹

The reason for using two different parsers is to take advantage of their unique strengths. `pdfplumber` is effective at extracting the original colors from the PDF, which is crucial for maintaining accurate placement and alignment in the annotations. In contrast, `PyMuPDF` converts all colors to sRGB, a format that sometimes blends similar colors, which can lead to slight inaccuracies in recognizing distinct colors.

However, `PyMuPDF` excels at extracting detailed font information more precisely than `pdfplumber`. To combine the strengths of both parsers, they are used

sequentially: `pdfplumber` first identifies and extracts colors and their positions, and then `PyMuPDF` uses this positional information to accurately extract font styles, sizes, and attributes such as italic, bold, superscript, and subscript. This sequential use of both parsers ensures that we capture comprehensive details about both the colors and the textual elements of the PDF.

The annotations and output files generated by the L^AT_EX Rainbow framework can be merged to create what are accessibility annotations. They can be seamlessly converted into various markup languages, such as HTML or Markdown, to facilitate wider accessibility and ease of use.

Accessibility annotation aligns with PDF, L^AT_EX, and Markup languages through element positions. This alignment ensures that the annotations are accurately reflected across different formats, enhancing the document's accessibility and maintaining consistency across various platforms.

3.3 Standardization of annotations

After I got the Markdown or HTML form of the PDF accessibility markup, I encountered an issue with some math formulas not being correctly displayed by browser-based math rendering libraries like `MathJax` [3]. This was primarily due to the persistence of custom math symbols defined using `\def` commands that were not adequately transformed into their corresponding Markdown forms, despite the earlier cleanup of the L^AT_EX code mentioned in Section 3.1.

To address this challenge, I utilized L^AT_EXML [13], a powerful tool designed to parse and convert L^AT_EX code into a markup language. It ensures that the L^AT_EX formula code is not only transformed into a format compatible with web standards but also that it unifies the styles of mathematical symbols sourced from various L^AT_EX packages.

I reassembled the annotations extracted from the PDF file into a `.tex` file.

```
\documentclass{article}
\input{preamble} % read from main.tex
\begin{document}
\section{C1B5E0}
$y = x + 1\$ % the first math formula
\section{1B3810}
$y = x + 2\$ % the second math formula
...
\end{document}
```

This file loads the preamble of the paper's source, and then puts all the mathematical formulas of a paper into separate sections, with the section's title

¹ github.com/pymupdf/PyMuPDF

being its hexadecimal RGB color code in the accessibility annotation.

By using `LATEXML` to convert the assembled `LATEX` code into HTML, I parsed the generated HTML document using Beautiful Soup, a robust HTML parser. This allowed me to navigate through the HTML structure efficiently and identify the sections containing mathematical formulas. The final step in my process involved replacing the original mathematical formula code within the accessibility annotations with the standardized code generated by `LATEXML`.

After the above three steps, we obtain a PDF file, an auxiliary file to record the label of each element on the PDF file, and its reading order, and an auxiliary file to record the tree structure of the document.

These auxiliary files help me to transform a PDF file, or any page, or any one of the chapters, into a markup language such as HTML or Markdown.

4 Serving as a dataset-making pipeline

According to arXiv, 90% of their submissions include `LATEX` source code [2], and each submission is accompanied by a PDF. The remaining 10% are available only as PDFs and might lack accessibility tags due to the absence of `LATEX` sources. Given the necessity for all scientific publications to be accessible, there's a need to derive accessibility tags directly from the PDF files. While some existing API² services offer this capability, they often produce significant mislabeling and noise.

To address these limitations, recent advancements in machine learning, particularly vision language models, have introduced document-specific solutions. An example of such innovation is the optical character recognition (OCR) model named Nougat [1], which utilizes a transformer architecture. Nougat processes screenshots of academic papers and converts them into Markdown. Importantly, it integrates mathematical formulas and tables within the page by converting them into `LATEX` code in the Markdown output, showcasing a significant step forward in document processing technology. Converted Markdown also provide accessibility to PDF.

One challenge with machine learning models, including Nougat, is their lack of precision. Nougat's accuracy issues may stem from inadequately detailed training data. Its training process aligns entire PDF pages to Markdown, which is less precise than aligning based on specific page element positions. This page-based alignment and Nougat's method of pro-

cessing one word at a time can lead to errors in recognizing the location and reading order of subsequent words as the OCR task progresses.

The effectiveness of machine learning depends heavily on the quality of the training data; the model needs detailed and high-quality data to extract sufficient features that enhance its performance. My method, which generates auxiliary files to record precise element positions and accessibility markers in PDFs, could serve as an invaluable resource for creating enhanced training datasets. This could significantly improve the performance of machine learning models like Nougat by providing them with more accurate and fine-grained data on text positioning and structure. My approach could help bridge the gap between current OCR capabilities and the demands for higher accessibility and accuracy in document processing.

5 Comparison to `tagpdf` and `SyncTEX`

`tagpdf` [5, 15, 16] is a `LATEX` package designed to facilitate the creation of PDF/UA-compliant accessible PDFs by providing core commands for tagging within `LATEX`. This approach allows updates directly to the `LATEX` kernel, avoiding the complexities associated with external patches. Highlighted at various `TEX` conferences, `tagpdf` addresses a gap in tools for experimenting with PDF tagging and accessibility. It introduces several enhancements to the `LATEX` kernel, including new PDF management features, automatic markup of paragraphs, and refined handling of page elements. `tagpdf` can be accessed through the “test-phase” key in the `latex-lab` package, allowing users to implement these features during the developmental stages of their documents.

`SyncTEX` [9] is a utility integrated into a `TEX` engine. It enhances the workflow between text editors and output viewers by providing synchronization capabilities, allowing seamless navigation between source code and the output PDF. `SyncTEX` generates an auxiliary file, which applications use to synchronize the text within the editor with the corresponding location in the PDF file.

My method, similar to `SyncTEX`, generates several auxiliary files that record coordinates corresponding to the positions of elements within a PDF. It also captures accessibility markers, including tags for elements, the reading order, and the expression of formulas. These elements are not embedded directly into the PDF, setting my approach apart from `tagpdf`. This strategy serves as a practical temporary solution, providing some of the functionality of `tagpdf` while it is still in the experimental stage.

² developer.adobe.com/document-services/apis/pdf-accessibility-auto-tag/



Figure 4: Labels of each crater dataset tile.

as a testing set while the remainder is used as a training set with the resulting F1-Score averaged together.

Our results are shown in Table 1. The scores were obtained from the respective papers with the exception of “Urbach ’09” which was obtained from [1] where it is used as a baseline comparison. Our CNN approach is

Figure 2: Example of coloring not working. The Figure has the caption label “Figure 4”, the reference number in Table “1”, and the citation “[1]”. They are all kept black because these elements are not colorable in the annotated L^AT_EX code.

6 Future work

6.1 Alignment without coloring

This work bridges PDF and L^AT_EX and Markup using color: the unique colors of each element. However, utilizing color also implies that it occupies a channel within the PDF output, leading to specific challenges.

First, there are instances where elements initially assigned a specific color by the author are overwritten in my process, resulting in the loss of original color information. Second, some elements, such as hyperlinks or caption labels in figures and tables (as depicted in Figure 2), derive their colors from package-level definitions rather than directly from the user’s L^AT_EX source code. For example, the \url command standardizes hyperlink colors across the document, which precludes assigning unique colors to individual links. Similarly, captions of figures and tables typically do not allow for unique coloring.

To address these issues, my forthcoming work will explore alternative methods beyond using color as a markup channel. Specifically, I plan to employ other embeddable features within the PDF, such as the tagging capabilities offered by the tagpdf package, as discussed in Section 5. By manually writing these tags, I aim to preserve the distinctiveness of document elements without overriding the original color assignments.

6.2 Parsing L^AT_EX code with T_EX engines

Despite meticulous efforts to parse user intent in their writing, a significant portion of papers from the arXiv remains under-annotated. My assumption was that every well-structured paper would include essential elements such as a title, author information, address, section titles, and body text. However, in practice, approximately 40% of papers lack annotations for at

least one of these components. The most commonly missing annotations are those for authors, titles, and abstracts, often due to the use of customized style files that obfuscate or alter standard formatting.

In Section 3.1, I discuss how user-defined macros have been partially managed using the de-macro package. However, numerous style files from journals and conference proceedings introduce additional commands, complicating the parsing process for the Python-based parser pylatexenc. The flexibility of L^AT_EX, attributed to its Turing-completeness [6], particularly challenges pylatexenc due to the prevalent use of the \def commands and \if conditions in style files, rendering the parser ineffective.

A T_EX engine, which must parse the source file during compilation, provides tracing options like \tracingmacros=1 that help humans understand how the T_EX engine expands custom commands. This tracing is detailed through logs that elucidate the functioning of various packages. There are L^AT_EX packages available to assist users in simplifying the log to make understanding the expansion of macros easier [10, 14]. Building on this, my planned approach involves enhancing the Python parser to interpret these logs. By leveraging the T_EX engine’s capabilities through log analysis, the parser is then expected to construct and interpret abstract syntax trees more accurately.

6.3 expl3 in L^AT_EXML

Section 3.3 discusses how I utilized L^AT_EXML to standardize mathematical formulas and tables within the paper code. This standardization process can be notably time-consuming. If expl3 is not included in the preamble, this conversion only takes a few seconds. However, if expl3 is loaded in a paper’s preamble, converting the L^AT_EX source code into HTML using L^AT_EXML can take over 20 minutes with T_EX Live 2024, or 10 minutes with T_EX Live 2021. The increased processing time can be attributed to the necessity for L^AT_EXML to load the entire expl3 package during each conversion, a package that has seen significant expansion in recent years due to active development. The L^AT_EXML development team has acknowledged this issue and is considering solutions such as caching expl3.sty or rewriting L^AT_EXML in Rust to improve efficiency.³

In future work, I plan to evaluate the potential impact of disabling the expl3 package loading on the standardization process. I anticipate minimal impact, as the standardization primarily depends on packages

³ github.com/brucemiller/LaTeXML/issues/2268

related to mathematics and table formatting rather than the `expl3` package.

7 Conclusion

In this paper, I introduce a method for enhancing the accessibility of PDF files that are compiled from L^AT_EX sources. This approach leverages coloring techniques to generate accessibility annotations and to align content across L^AT_EX, PDF, and various markup languages. Currently, my method applies exclusively to scientific papers available on arXiv that include L^AT_EX source code. However, it also serves a broader purpose by facilitating the creation of datasets. These datasets can be utilized to train machine learning models, which can improve the generation of accessibility annotations for scientific papers where only the PDF versions are available. This development holds the potential for increasing the accessibility of scientific literature.

Acknowledgement

This work was conducted within the research project InsightsNet (insightsnet.org) which is funded by the Federal Ministry of Education and Research (BMBF) under grant no. 01UG2130A.

References

- [1] L. Blecher, G. Cucurull, et al. Nougat: Neural optical understanding for academic documents, 2023. arxiv.org/abs/2308.13418.
- [2] S. Brinn, C. Cameron, et al. A framework for improving the accessibility of research papers on arxiv.org, 2024. arxiv.org/abs/2212.07286.
- [3] D. Cervone. MathJax: a platform for mathematics on the web. *Notices of the AMS*, 59(2):312–316, 2012.
- [4] C. Duan, Z. Tan, S. Bartsch. LaTeX rainbow: Universal LaTeX to PDF document semantic & layout annotation framework. In *Proceedings of the Second Workshop on Information Extraction from Scientific Publications*, T. Ghosal, F. Grezes, et al., eds., pp. 56–67, Bali, Indonesia, Nov. 2023. Association for Computational Linguistics. doi.org/10.18653/v1/2023.wiesp-1.8
- [5] 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
- [6] A.M. Greene. BaStX: An interpreter written in T_EX. *TUGboat* 11(3):381–392, Sept. 1990. tug.org/TUGboat/tb11-3/tb29greene.pdf
- [7] P. Gács. de-macro—Expand private macros in a document, Dec. 2020. ctan.org/pkg/de-macro
- [8] ISO Central Secretary. Document management applications — electronic document file format enhancement for accessibility. Standard ISO 14289-2:2024, International Organization for Standardization, Geneva, CH, 2024. www.iso.org/standard/82278.html
- [9] J. Laurens. Direct and reverse synchronization with SyncT_EX. *TUGboat* 29(3):365–371, 2008. tug.org/TUGboat/tb29-3/tb93laurens.pdf
- [10] B. Le Floch. unravel: Watching T_EX digest tokens, Jan. 2024. ctan.org/pkg/unravel
- [11] M. Li, Y. Xu, et al. DocBank: A benchmark dataset for document layout analysis. In *Proceedings of the 28th International Conference on Computational Linguistics*, D. Scott, N. Bel, C. Zong, eds., pp. 949–960, Barcelona, Spain (Online), Dec. 2020. International Committee on Computational Linguistics. doi.org/10.18653/v1/2020.coling-main.82
- [12] J. MacFarlane, A. Krewinkel, J. Rosenthal. Pandoc. github.com/jgm/pandoc
- [13] B. Miller. L^AT_EXML: A L^AT_EX to XML/HTML/MathML Converter, Feb. 2024. math.nist.gov/~BMiller/LaTeXML/
- [14] F. Mittelbach. The trace package. *TUGboat* 22(1/2):93–99, Mar. 2001. tug.org/TUGboat/tb22-1-2/tb70mitt.pdf
- [15] F. Mittelbach, C. Rowley. L^AT_EX Tagged PDF—a blueprint for a large project. *TUGboat* 41(3):292–298, 2020. doi.org/10.47397/tb/41-3/tb129mitt-tagpdf
- [16] C. Rowley, U. Fischer, F. Mittelbach. Accessibility in the L^AT_EX kernel—experiments in Tagged PDF. *TUGboat* 40(2):157–158, 2019. tug.org/TUGboat/tb40-2/tb125rowley-tagpdf.pdf
- [17] J. Singer-Vine, The pdfplumber contributors. pdfplumber, July 2024. github.com/jsvine/pdfplumber

◊ Changxu Duan
 Technische Universität Darmstadt
 Residenzschloss 1
 64283 Darmstadt
 Germany
 changxu.duan (at) tu-darmstadt dot de
 ORCID 0000-0003-0547-0901

TeX (Live) and accessibility at arXiv

Norbert Preining

Abstract

This article combines two talks at the TUG 2024 conference, one about TeX (Live) at arXiv, and one about accessibility and HTML papers. We will give a short introduction of what arXiv is and its importance for open science. After this introduction, the first part deals with how we use TeX at arXiv, followed by a second part on improving accessibility at arXiv.

1 Introduction

arXiv is the world's largest and oldest scientific preprint server, and a champion of open science. Started in 1991, arXiv presently holds more than 2.4 million articles and is growing at an ever-increasing rate.

In many areas of physics, math, and computer science, cutting edge research is first made available on arXiv. Examples are

- LLM research (OpenAI, Deepmind, etc.)
- LIGO (Gravity wave research; 2017 Nobel Prize in physics)
- Proofs of famous theorems (Grigori Perelman)

And while most papers posted to arXiv are eventually published in journals, in some fields research is often made available only on arXiv. Even for work subsequently published in journals, early posting to arXiv enables scientists to more rapidly incorporate shared results, and assert prior authorship.

What sets arXiv apart from many other services is *openness*: All articles are freely available without any paywall, and in addition, for more than 90% of the articles, sources are available—big thanks to Paul Ginsparg et al. for insisting from the start in 1991 that scientists submit the source code for their papers!

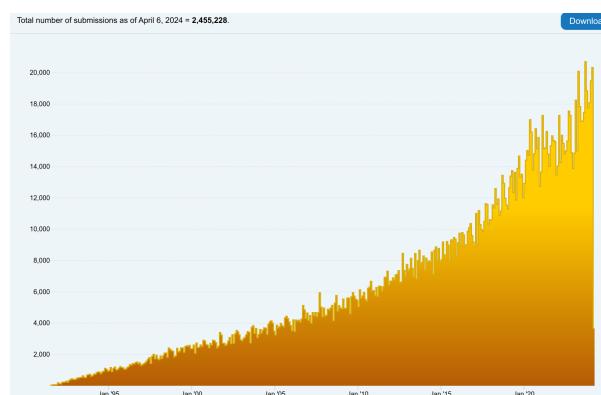


Figure 1: arXiv monthly submissions, 1991–present

arXiv also profits from and contributes to open source software: Our fundamental tools include TeX, nowadays TeX Live, and L^AT_EXML for the conversion to HTML. Both are open source projects, and we are in very close contact with both maintainer teams.

2 TeX at arXiv

About 90% of submissions are in (L^A)TeX, going back more than 30 years. For the submissions where we have TeX source, we consider the PDF artifacts as generated and it is sometimes necessary to recreate them. Consequences of this approach is that we have to keep old versions of TeX available to ensure compilability even of documents from the start of arXiv. This means that, as of now we are having available (and regularly use for PDF rebuilds) the following TeX installations:

- teTeX 2 and 3
- TeX Live: 2009, 2010, 2011, 2016, 2020, 2023

In particular teTeX 2 has put me in front of a few challenges, since the first teTeX I (ab)used heavily when I started maintaining TeX in Debian was teTeX 3, which has a very different configuration approach than teTeX 2.

2.1 AutoTeX

Up to now, the system that accepts a submission, and converts it to a PDF, is a Perl program called AutoTeX [6], written at arXiv. It has been maintained since the beginning of arXiv, but hasn't changed a lot since then. Its main jobs are detecting the format of the submission (plain TeX or L^ATeX) by running respective engines and accepting the first one that succeeds in compiling the submission. This already can lead to strange cases where a file that is not the intended top level document is nevertheless considered as such and compiled using plain TeX, despite being a L^ATeX document.

AutoTeX supports only two formats: L^ATeX with dvips/ps2pdf or with pdflatex, and plain TeX with dvips/ps2pdf route. For L^ATeX calls, AutoTeX reruns several times until references are stable. For plain TeX files only one run is executed.

In the end, all generated and additional PDF files are combined into a final PDF using pdfpages.

There are several shortcomings with this approach:

- No support for LuaTeX or XeTeX or any other engine
- No support for BIBTeX or biber/BIBL^ATeX; bbl files must be uploaded
- No support for makeindex

- Merging with pdffpages breaks hyperlinks in documents
- Detection of main document is often incorrect (conference templates are often uploaded together with actual paper)
- arXiv watermarking is fragile
- ... (probably so many other problems I'm not remembering or haven't seen yet)

2.2 The (very near) future

arXiv is currently in a complete restructuring process, modernizing practically all parts of the system, and moving core parts to a cloud-based system.

The new \TeX backend system will be based on a dockerized \TeX to PDF conversion system, which allows for a certain restricted set of parameters (usage of additional trees, watermarking, etc.). Older systems (which we have to keep available to re-compile older submissions) will be dockerized together with AutoTeX. The system will contain an auto-dispatch component to the respective \TeX Live year container depending on submission date.

Advantages we get from leaving AutoTeX behind and rewriting include, first and foremost, *less magic*—getting rid of the *auto* part of AutoTeX and replacing it with clear indications by the submitter which file should be compiled with which engine. The frontend submission workflow will see considerable changes in the very near future, too.

Another great advantage of the new system will be the possibility to build *reproducible documents* [5]—that is, documents that can be bitwise compared after changes of system components.

Further improvements include better watermarking (by switching to the `pymupdf` library for watermarking), and better PDF composition (using `gs` which preserves hyperlinks; see [3] for a related discussion).

We also will stop providing additional class and style files, which many users have relied on in the past. Going forward, we expect submissions to contain class files for journals that are not provided by \TeX Live itself. The problem is with class and style files distributed by certain journals that have restrictions in place making it impossible to include them in \TeX Live. At arXiv, we have provided the latest version of these class files for many years, but the proliferation of more and more non-TL files, and the management of versions of those files, has created a considerable burden at arXiv.

We have thus decided that going forward, only files distributed by \TeX Live itself will be automatically available, and all other files need to be included in the submission.

2.3 The (hopefully near) future

With the basic rewrite already done, we are now aiming at including long requested features in the (hopefully) near future. At the top of that list is an improvement for the submission process itself, making it easy and painless for *easy* submissions, but still giving the user full freedom to submit a complicated arrangement within a submission.

Also very high up on our list is support for Lua\TeX and Xe\TeX , as well as for bibliographies and indices (no need to ship pre-made `.bb1` files!).

Also in the pipeline is publishing our \TeX to PDF docker web API service as open source.

3 Accessibility at arXiv

This is an abridged version of an article at the DEIMS 2024 conference [4]. The authors of the DEIMS paper and the full arXiv team are responsible for all the achievements, I am only reporting on the current status!

arXiv has a mandate to continuously improve access to scientific research, and our long-term mission is simply to serve the needs of the scientific community through openness, collaboration and scholarship. Everyone has the right to participate in the wealth of scientific knowledge contributed to arXiv by researchers from all over the world. Accessibility is inherent to our mission of championing open science. When we asked scientists with disabilities how arXiv could help make research more accessible they told us: add HTML as a format for papers.

Over the past few years, arXiv has made good progress in making our website [2] more accessible according to W3C WAI guidelines. While this allows people with disabilities to more easily find and access papers, they often cannot read them because arXiv's papers are available almost exclusively in PDF format, which has low native accessibility.

What we heard from scientists with disabilities, standards experts, and accessibility researchers is that PDF will always be playing catch up with HTML when it comes to accessibility. Adding HTML as a format on arXiv will get us closer to fulfilling the promise of open science.

3.1 PDF limitations wrt accessibility

PDF has been designed as a *page description* language, representing the physical page to be printed. It is an excellent format for this purpose, but the internal representation poses a lot of problems when it comes to accessibility.

Page and reflow paper geometry and the actual screen dimensions are different in most of the cases,

and zooming in often requires horizontal scrolling. Although there are moves by Adobe to provide some kind of responsive design, i.e. “Liquid Mode”, this is mobile-only and proprietary.

Structural limitations When a page is described in PDF, lots of semantic information is lost (at least until PDF/UA2): headers, captions, all semantic entities are reduced to purely typographic elements. While “Tagged PDF”, introduced by Adobe, aims at improving the situation, most documents out now are not properly tagged, and there is very poor support for the creation of tagged PDFs.

Recent developments on the L^AT_EX kernel side show promising advances, but it cannot deliver now a solution to the scale of the arXiv corpus, since documents still require manual work to achieve proper tagging.

3.2 HTML is a better solution

HTML already provides now what most of the PDF and PDF/UA2 is trying to deliver in the future: responsive design, dark mode, built-in language translations, add-ins for, e.g. dyslexia or visually impaired, all backed by a rich marketplace of assistive technologies.

The HTML code also preserves the semantic structure and intent of the document, allowing for better representation in e.g. screen readers.

Last but not least, text harvesting, e.g. for LLM, is easier when based on HTML, while text extraction based on PDF can get rather tricky.

But there are stumbling blocks: online scientific work is mostly available only in PDF format, and conversion from PDF to HTML is challenging, in spite of some interesting work that `allen.ai` has done in this area [1], because structure, once lost, is difficult to reconstruct.

3.3 Converting L^AT_EX to HTML

Since more than 90% of the arXiv’s submissions are in T_EX, and lately mostly in L^AT_EX, it is natural to consider direct L^AT_EX to HTML conversions. Since T_EX itself also produces typographic information where all structure is lost, relying directly on the T_EX engine to provide HTML output is non-trivial. Thus, all available converters basically operate in the same way, namely providing XML/SGML/HTML renderings for each and every command available. With T_EX4ht, T_EX itself is used, while L^AT_EXML uses Perl. For all systems it remains an immense project to provide XML renderings for each command defined in each add-on package in the T_EX and L^AT_EX landscape. Thus, all of the solutions will remain partial for the foreseeable future.

The main solutions currently available are L^AT_EXML maintained by Bruce Miller and Deyan Ginev at NIST (National Institute of Standards and Technology)

T_EX4ht created by Eitan M. Gurari, now maintained by Michal Hoftich

L^AT_EX support for tagged PDF: L^AT_EX core team, early stage

arXiv took a pragmatic approach and investigated the existing tools. T_EX4ht and L^AT_EXML were roughly tied in the quality of the HTML produced, but L^AT_EXML was found to have a larger library of supported packages and better ongoing support. Beyond that, the predecessor project *ar5iv* already used L^AT_EXML.

The *ar5iv project* was started by Dr. Michael Kohlhase from KWARC and Ph.D. student Deyan Ginev. Its intent was to offer HTML versions for arXiv’s entire L^AT_EX corpus, using L^AT_EXML. It had about 20% failed conversions, as well as other visual glitches, but saw significant improvements over the years. Unfortunately, reconversion of old articles is a costly endeavor; on Google Cloud with an approximate average cost of \$0.015 per article, it would amount to approximately \$30,000 for the entire arXiv corpus.

Further pain points we face with the L^AT_EXML conversion is the already mentioned long tail of less common packages that are not supported, as well as author-written macros and extensions (here T_EX4ht has the advantage of using T_EX itself for macro expansion). Furthermore, there will always remain some edge cases where L^AT_EX constructs don’t render correctly.

Despite all the abovementioned shortcomings, providing HTML pages—even with glitches—has proven a resounding success with the accessibility community, because even in the presence of those glitches, the papers remain generally readable, in particular for screen readers. To put it in simple words:

Something is better than nothing!

3.4 Rollout and user interface

An HTML version of arXiv’s corpus has been available for several years now in the *ar5iv* project, but it remains less known (compared to arXiv itself). Thus, bringing the *ar5iv* project “in house” and providing the HTML versions directly alongside the PDF version made the change much more visible and profound.

In addition to the delivery of the HTML version, we have already included the HTML generation into the submission process, asking authors to review

The screenshot shows the arXiv article view for arXiv:2404.00468. The page title is "On P=NP Either False or Independent of ZFC". The "Access Paper:" section includes links for "View PDF", "HTML (experimental)", "TeX Source", and "Other Formats". Below this is the "Current browse context: math.LO" section, followed by "References & Citations" and "Export BibTeX Citation". At the bottom, there's a "Submission history" section.

Figure 2: Article view with HTML entry

the HTML version in addition to the PDF version. While a failed generation of the PDF version blocks further processing, we do not consider a failed HTML conversion as a blocker. We do hope that authors will review the HTML rendering and consider making adjustments to their L^AT_EX source to improve the HTML version.

The HTML versions also prominently feature a feedback button that allows users to indicate incorrect renderings and other problems with the conversions. With millions of users performing QA on our HTML—we already have seen thousands of reports—we feel we can improve the conversion process in the future.

In the article view on arXiv (fig. 2), we have added an additional format entry, as well as a “Beta” label. As of now, the HTML format button will only be shown with new submissions until we backfill the historical corpus over time.

When visiting the HTML version of a paper (fig. 3), and if the L^AT_EXML conversion issued warnings, these warnings are shown in a separate panel at the top. This will often be the case for packages used in the L^AT_EX code that are not supported at present. In the screenshot we see the L^AT_EX package *inconsolata* not being supported.

The next screenshot (fig. 4) shows that formulas and tables are supported, included graphics are shown as is, and that a dark mode setting in the browser is taken into account. Mind also the prominent “Report issue” button (lower right corner)!

3.5 Future work and summary

Making our corpus accessible is an open-ended project, and we are aware of the shortcomings our current solutions have, but we are also aware of the profound positive impact the addition of HTML versions has already had in the accessibility community.

The screenshot shows the arXiv article view for arXiv:2404.00468. A red warning panel at the top states: "This is experimental HTML to improve accessibility. We invite you to report rendering errors. Learn more about this project and help improve conversions." It includes a "Report Issue" button. The main content area shows the abstract and sections 1 through 3 of the paper.

On P=NP either false or independent of ZFC

Figure 3: HTML version view with warning panel

The screenshot shows the arXiv article view for arXiv:2404.00468. The warning panel at the top is still present. The main content area contains mathematical definitions and formulas, such as Definition 3.1 and Definition 3.2, along with their proofs. There are also sections for "Image" and "Report issue".

Figure 4: HTML version view with math and report issue button

To pick a few items from the long list of future work:

- Continue to work with the L^AT_EXML team to improve conversion process
- Figure out a cost-effective way to periodically re-compile the whole corpus to pick up these improvements
- Revisit tooling in a few years when the L^AT_E team is further along—we are hopeful that the work to produce tagged PDFs will also enable the generation of HTML output
- Make charts and graphs more accessible

- Possibly provide a way for users to access the data behind graphs
- Auto-caption images and graphs, via AI or crowd sourcing or both.

To summarize, we want to stress that accessibility of scientific documents is an important improvement, and the feedback from the community and disabled scientists has been overwhelmingly positive—HTML even with glitches is better than PDF! And while we are still far from “the last mile”, we are making great progress thanks to a great community and open source support.

3.6 Acknowledgments

We thank the many scientists with disabilities who so generously shared their expertise, insights, and feedback, and guided arXiv’s efforts towards impact.

This material is based upon work supported by the National Science Foundation under Award No. OAC-2311521. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Big thanks to the whole arXiv team—I have only recently joined and am reporting on **their** achievements!

References

- [1] Allen Institute for AI. Paper to HTML. papertohtml.org/
- [2] arXiv project. arxiv.org
- [3] K. Berry. Production notes. *TUGboat* 45(1):154–154, 2024. doi.org/10.47397/tb/45-1/tb139prod
- [4] C. Frankston, J. Godfrey, et al. HTML papers on arXiv—why it is important, and how we made it happen, 2024. arxiv.org/abs/2402.08954
- [5] Reproducible builds website. reproducible-builds.org
- [6] T. Schwander. AutoTeX on CPAN. metacpan.org/pod/TeX::AutoTeX

◊ Norbert Preining
arXiv / Cornell University
norbert (at) arxiv dot org
<https://arxiv.org/>

Legal and cultural landscape of mathematics accessibility in the United States: 2024

Jeffrey Kuan

In April 2024, the United States Department of Justice released new guidelines, which will mandate that all state and local institutions of higher learning comply with WCAG2.1AA standards by either April 2026 or April 2027. This survey will briefly summarize the context of these guidelines, and how they affect mathematics accessibility in the United States.

Disclaimer: this paper should not be construed as legal advice.

1 Introduction

In the United States, accessibility (for persons with disabilities) has many legal requirements. For the purposes of this paper, I will quote the National Center on Accessible Educational Materials [1] for a definition of accessibility:

“... accessibility really is individualized. In fact, according to the Office for Civil Rights, accessibility is happening anytime a person with a disability can acquire the same information, engage in the same interactions, and enjoy the same services as a person without a disability in an equally effective, equally integrated manner, and with substantial equivalent ease of use.”

From this definition, one may immediately note that it is not possible for educational material to be “100% accessible”, due to different individuals having different needs. At the same time, however, it must be possible for educational material to be compliant with legal requirements that protect the civil rights of persons with disabilities. Otherwise, there would exist a law that would be impossible to follow.

A common set of accessibility guidelines are the Web Content Accessibility Guidelines (WCAG) published by the World Wide Web Consortium (W3C). These technical guidelines are often used as a benchmark for accessibility in a legal context. Most recently, announced Department of Justice rules have mandated WCAG2.1AA standards for all state and local entities in the United States, including public universities and colleges. This survey will briefly discuss the context of these rules, as well as its expected impact on accessibility in mathematics education and research.

2 WCAG

Shortly after the founding of the W3C in 1994, the first web accessibility guidelines began to be developed in 1995. The WCAG have undergone several

versions, with WCAG3.0 under development and version 2 being current. Each version has three levels, denoted A, AA and AAA, with AAA levels including all AA levels, and AA levels including all A levels. The highest level, AAA, is generally considered difficult to follow. For instance, success criterion 1.2.6 (“sign language interpretation is provided for all pre-recorded audio content in synchronized media”) is under level AAA and usually is not met, with the movie *Barbie* being a notable exception.

The WCAG2.0AA standards contain 38 success criteria, while WCAG2.1AA includes an additional 12. Notable success criteria are 1.3.4 (orientation) and 1.4.12 (text spacing). It is beyond the scope of this paper to delve into the specifics of all the criteria, but I will note that the guidelines are summarized under the acronym POUR (Perceivable, Operable, Understandable, Robust), corresponding to the first digit in each success criterion.

The timeline of the WCAG release dates fits awkwardly in the history of American laws and regulations (which is unsurprising, given that W3C is an international organization). The most influential federal civil rights legislations, the Rehabilitation Act of 1973 and the Americans with Disabilities Act of 1990, were both passed before the development of the internet. Next, I will summarize the various laws passed prior to the Department of Justice rules.

3 Federal laws

3.1 Rehabilitation Act of 1973

The Rehabilitation Act of 1973 is a United States federal law protecting the civil rights of people with disabilities in the federal sector. For most American mathematicians, the most relevant section of that act is Section 504:

“No otherwise qualified individual with a disability in the United States . . . shall, solely by reason of her or his disability, be excluded from the participation in, be denied the benefits of, or be subjected to discrimination under any program or activity receiving federal financial assistance or under any program or activity conducted by any Executive agency . . .”

As a large portion of mathematical research programs receive federal financial assistance from the National Science Foundation, such programs are under the jurisdiction of section 504.

In 1998, Congress passed an amendment to the Rehabilitation Act, titled section 508. According to the section 508 webpage [5]:

In 1998, Congress amended the Rehabilitation Act of 1973 to require federal agencies to make their electronic and information technology (EIT) accessible to people with disabilities. The law . . . applies to all federal agencies when they develop, procure, maintain, or use electronic and information technology. Under Section 508, agencies must give disabled employees and members of the public access to information comparable to the access available to others.

Beginning in January 2018, the U.S. Access Board has required WCAG2.0 standards for Section 508. Only a few months later, W3C released WCAG2.1 standards in June 2018.

3.2 Americans with Disabilities Act of 1990

The Americans with Disabilities Act of 1990, or ADA, is a civil rights law that prohibits discrimination based on disability. For colleges and universities, the most relevant parts are:

- Title II of the ADA requires “state and local governments to make sure that their services, programs, and activities are accessible to people with disabilities.”
- Title III of the ADA “prohibits discrimination on the basis of disability in the activities of places of public accommodation.”

For students at American colleges and universities, the distinction between public and private universities usually is irrelevant, although they are technically governed under different titles of the ADA. Title I of the ADA prohibits employment discrimination, but this title affects faculty and staff more than students. Unless the reader is currently involved in an employment dispute at their college or university, it is less likely that Title I is relevant to them.

4 Recent developments

In more recent years, there have been several legal developments relevant to accessibility in mathematics.

4.1 State laws

Following the 2018 rule that section 508 use WCAG2.0 standards, many states passed relevant laws.

- In the state of Texas (where the author is currently employed), Texas Administrative Code section 206.70 requires all new and changed websites to meet WCAG2.0 Level AA (excluding Guideline 1.2 Time Based Media), in reference to Section 508; effective April 18, 2020. A

more educational-specific provision is in section 213.39, which states that the president or chancellor of each institution of higher education shall ensure appropriate staff receives training necessary to meet accessibility-related rules.

- California Assembly Bill 434 requires state websites (including institutes of higher education) to comply with WCAG2.1AA by July 1, 2019, again referencing Section 508. The law was signed in 2017. It should not be confused with California Assembly Bill 1757, which applies to businesses subject to California's Unruh Civil Rights Act.
- Colorado House Bill 21-1110 requires all web content, including internal content, produced by institutions of higher education to be WCAG2.1AA compliant. A separate house bill, HB24-1360, created a Disability Opportunity Office to support residents with disabilities.

Even from these three examples, one can notice the piecemeal nature of accessibility laws for public institutes of higher education.

4.2 Captioning of videos

In 2015, the National Association of the Deaf filed legal cases against Harvard and MIT, alleging that their publicly posted course content violated section 504 of the Rehabilitation Act and Title III of the ADA, because the captions were not sufficiently accurate. In 2019–2020, the cases were settled in favor of the National Association of the Deaf. During this same time, the University of California, Berkeley (a public institution, and therefore under Title II) removed 20,000 free videos in 2017 and placed them behind university login.

5 Department of Justice rules

In April 2024, the Department of Justice released new rules for the interpretation of Title II of the Americans with Disabilities Act in the context of web accessibility. The rules were posted following a time for open comments. Below, we highlight some of the new rules, as it relates to local and state institutions of higher education. Note that these rules do not apply to private institutions (and businesses), which are covered under Title III. All page numbers below refer to the page numbers in the final rule on the PDF posted on the Federal Register [4].

5.1 WCAG2.1AA compliance

According to the ADA factsheet:

Requirement: The Web Content Accessibility Guidelines (WCAG) Version 2.1, Level AA is the technical standard for state and local governments' web content and mobile apps.

Interestingly, this requirement includes “password-protected course content in elementary, secondary, and postsecondary schools” (page 31360). This is analogous to the Colorado law, and thus “hiding” educational content behind institutional passwords does not exempt it from the new rules.

5.2 Timeframe

Public institutions serving populations of less than 50,000 have until April 26, 2027 to comply with the rule. Public institutions serving populations of 50,000 or more have until April 26, 2026 to comply with the rule. Community or city colleges in small populations could have until 2027; every state college or university has two years.

5.3 PDF/UA-1

Because most mathematicians produce \TeX -generated PDFs, there is a unique interest in PDF accessibility standards. The Department of Justice did consider PDF/UA-1, but ultimately found that WCAG2.1AA would both “enhance” (page 31344) the accessibility of PDFs, while maintaining the “balance” with administrative costs (p. 31350). In part to address concerns that public entities would simply “remove” content, the Department of Justice pointed out (pages 31346–31347) that many states had already been using WCAG2.0AA, and therefore public entities were likely familiar with the 38 guidelines of WCAG2.0AA, needing only an additional 12 guidelines to meet WCAG2.1AA.

5.4 Conforming alternative versions

Under WCAG, a “conforming alternative version” is allowable. For example, a non-accessible PDF may be posted if the reader can access the same information and functionality via a MathML webpage. However, the Department of Justice now states (section 35.202, page 31382):

“... a public entity may use conforming alternate versions of web content ... only where it is not possible to make web content directly accessible due to technical or legal limitations.”

The question of legal issues related to intellectual property law (such as textbooks) is addressed briefly on page 31377, and offers little clarification.

5.5 Exceptions

There is a notable set of exceptions [3] to these rules. At this time, I am unwilling to publicly comment on these rules.

6 Implications for mathematics

Needless to say, mathematics provides unique challenges for accessibility. At this point, I can only conjecture on what will come next.

6.1 Technical restrictions for PDF

Due to the clause that conforming alternative versions are allowable only under technical or legal limitations, there will likely be legal disputes about the technical limitations of PDFs. Currently, there is a great deal of public confusion concerning accessibility of TeX-generated documents. For instance, a well-respected accessibility resource run by Penn State University [2] falsely claims that

“A PDF file created from a .tex file is always inaccessible.”

Perhaps as a result, an accessibility advocate at the arXiv Accessibility Forum in 2023 suggested banning PDFs as a file format. Likely this question will be litigated in 2026 or 2027.

6.2 Removal of material

Some research universities may follow Berkeley’s lead and remove all course content, while ordering grade inflation to compensate. Ultimately, this is legally allowable and difficult to fight without providing additional resources for accessibility.

6.3 Federal funding

More optimistically, some funding agencies have offered increased funding for accessibility. As just one example, I received \$1,860 in salary to improve the accessibility of the Texas A&M Math REU webpage (software was covered with my accessibility company, Tailor Swift Bot). Furthermore, some Department of Education funded projects, such as Ximera, have budgeted for accessibility.

6.4 Instructional designers

To support accessibility in mathematics, some colleges and universities may invest in instructional designers to support faculty. At Texas A&M during Spring 2024, there were no instructional designers to support the 18 departments in the College of Arts and Sciences, despite being the largest public university in the country. However, this is perhaps more specific to Texas A&M University.

6.5 AI

Some colleges and universities may believe that AI can automatically create accessible documents. However, given that this issue was already litigated in the National Association of the Deaf vs. Harvard and MIT, this is very unlikely to be legally allowable, given the current state of AI.

References

- [1] National Center on Accessible Educational Materials. Defining the term accessible. youtu.be/ojthp08tc0w
- [2] Penn State. Equation format and accessibility. accessibility.psu.edu/math/equations/
- [3] US Department of Justice. Fact sheet: New rule on the accessibility of web content and mobile apps provided by state and local governments. ada.gov/resources/2024-03-08-web-rule
- [4] US Department of Justice. Nondiscrimination on the basis of disability; accessibility of web information and services of state and local government entities. govinfo.gov/content/pkg/FR-2024-04-24/pdf/2024-07758.pdf
- [5] US General Services Administration. IT accessibility laws and policies. section508.gov/manage/laws-and-policies/

◊ Jeffrey Kuan
jkuan (at) tailorswiftbot dot com

Extending Peter Flynn's bookshelf package for multilanguage libraries

Boris Veytsman

Due to the COVID, TUG2020 was held online. Figure 1 shows the drawing for the conference by Jennifer Claudio. As befits a true artist, Jennifer manages to reproduce the Zeitgeist with a well-chosen detail: the stylized bookshelves. They were created with the *bookshelf* package [1], which was released during the pandemic. It was used by many of us to generate the backgrounds for remote meetings. These bookshelves remind one of the time of endless meetings, fear, loneliness, sickness and death.

Peter's package uses a clever algorithm to create interesting images, different for each TeX run. It takes a BIBTeX catalog of books (many electronic book managers, like *Calibre* [2], can export the book list in this format). For each book it performs the following steps:

1. Select a random rectangle size.
2. Select random foreground and background colors. If the contrast is too low, repeat.
3. Select a random font.
4. Typeset author and title to fit in the box.

The result for my electronic library is shown on Figure 2. (Grayscaled for print; online, you might like to zoom in to see the variety of colors and fonts used.)

Besides creation of backgrounds, this package may be used also for an amusing game, which is quite suitable for long boring remote meetings. Take a look at some spines (Figure 3). Can you guess which fonts



Figure 1: Jennifer Claudio's drawing for TUG2020

were used to typeset them? You may add a point for each correctly guessed font, and additional points for correctly guessed author or style. To check your answers you need to know that the little numbers after the books are actually the numbers of the fonts in the main list of fonts used by the package. For example, looking at the font numbers on Figure 3, we get:

- 589:** AvenirLTStd-Heavy
- 9541:** KyivTypeTitling-Bold2
- 2784:** Concourse4Italic, Stylistic Set 3
- 17266:** XITS-BoldItalic
- 68:** Alegreya-ExtraBoldItalic, Stylistic Set 4
- 13971:** Nunito-ExtraLight
- 16390:** SourceSansPro-Black, Small Caps
- 10119:** KyivTypeTitling-Bold, Stylistic Set 4
- 15373:** RobotoSerif-Black, Old Style Numbers
- 536:** Arsenal-Bold, Stylistic Set 2
- 1490:** BradleyDJR-Micro, Historical Ligatures

An astute reader might understand at this point that Figures 2 and 3 were not produced by the original version of Peter's package. The reason is that some of the books on Figure 3 have Cyrillic spines (Ukrainian and Russian, to be precise). The fonts used for these books (XITS, Alegreya, Nunito, Source-SansPro, KyivTypeTitling) contain Cyrillic glyphs. However, since the font selection is random (see item 3 in the algorithm above), we can get fonts incapable of typesetting the spines. Since the number of books with non-Latin scripts in my library is large, the probability of such events is close to 1.

At first I restricted the selection of fonts only to those that had both Latin and Cyrillic glyphs. However, there were fonts I liked to see on my shelf which did not have Cyrillic letters. Also, I wanted a solution suitable for libraries more versatile than mine, with books in Arabic, Hebrew, Malayalam, Sanskrit, etc. I wanted to be able to typeset any catalog with any number of languages, and use any suitable font.

One solution would be to use *Language* tag of the fonts: we can add this tag to each book, create separate pools of fonts for each script, and then randomly select a font from the given pool. However this would require manual tagging of each book and a rather complicated font selection algorithm, especially for the books with several scripts in the title. Therefore I decided to use the same logic Peter used for color selection: for the given book select a random font. If the book spine can be typeset with this font, use it, otherwise repeat selection.

To check whether we can typeset the given string with the given font we use the primitive `\iffontchar`.

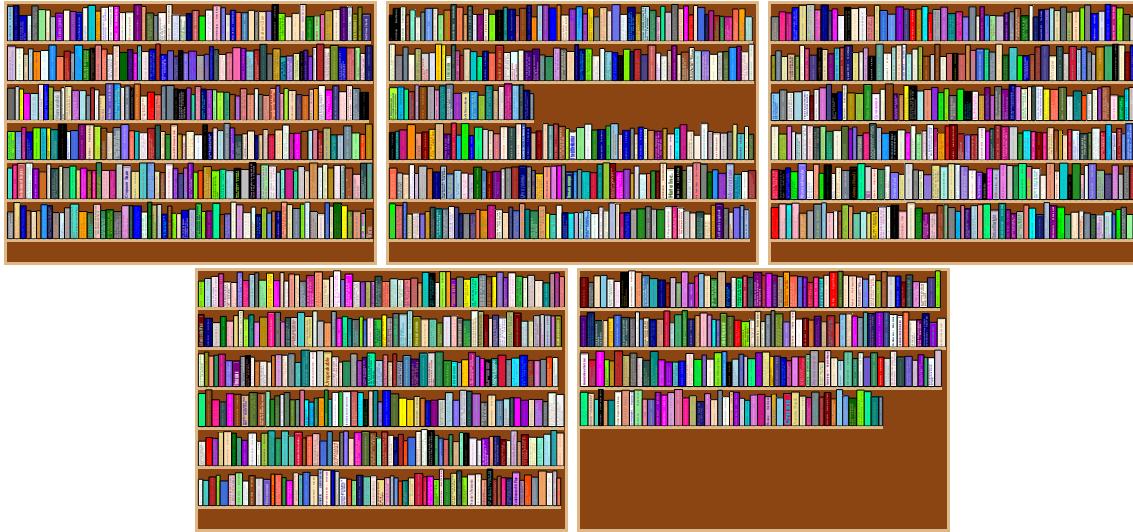


Figure 2: The author's electronic library

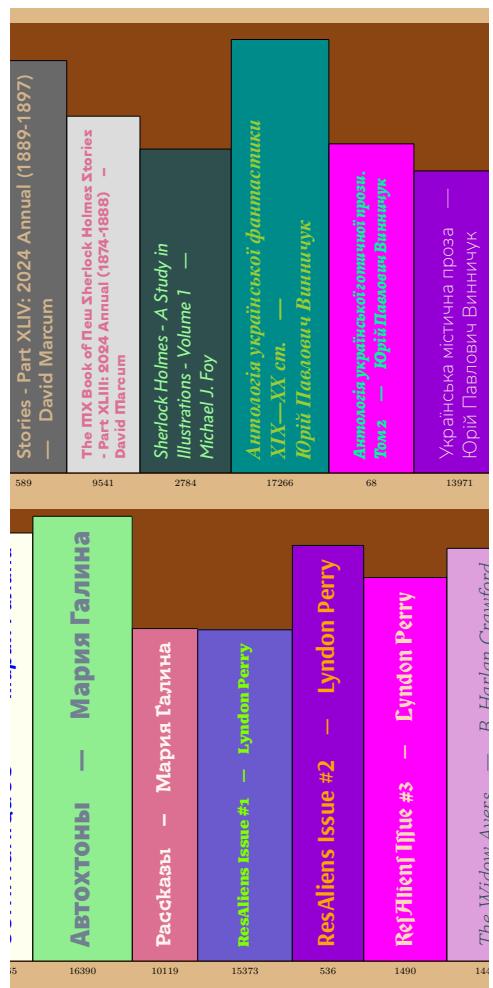


Figure 3: Several books from the author's library

The logic of the algorithm is straightforward: we map the primitive over the string, and bail out early if we find a character that cannot be typeset. The implementation is easier in `expl3` language; see Figure 4. This code defines a macro `\CanTypeSetTF{<string>}<{true}><{false}>`. It calls either `{<true>}` branch or `{<false>}` branch depending on the results of the typesetting test.

Since I wanted to demonstrate the possibilities of my fonts, I decided to change the source of them in the package. Both `XeTeX` and `LuaTeX` can use system fonts (those in the locations known to all applications on your machine), and `TEX` fonts (those known to your `TEX` installation). Peter's package can use any source, but the scripts provided with it get the list of fonts in the system directories. `TEX` Live has a very large collection of interesting fonts, to which I have added some that I've purchased or downloaded. Thus I decided to switch to the `TEx` fonts. I also wanted to demonstrate stylistic variants, swashes, old-style figures, so I wrote a script that lists these variants for the given font, as shown on Figure 5.

These changes lead to another problem. The number of fonts together with their variants turned out to be huge (19 183 on my machine). The trial-and-error algorithm for choosing a random font may open several fonts per book. A decent library (Figure 2 has 1584 books) probes many fonts from this list. Thus the package may want to open thousands of fonts for a single run. The number of fonts that a modern engine can open is much larger than in the old days, and can be further extended by changing the config file (I am grateful to Frank Mittelbach

```
\prg_new_if:NNn \__SIL_primitive_font_glyph_if_exists:n {TF,F}
{
  \tex_iffontchar:D \l_fonts_spec_font '#1 \scan_stop:
    \prg_return_true:
  \else:
    \prg_return_false:
  \fi:
}

\prg_new_if:NNn \__SIL_can_typeset:n {TF}
{
  \typeout{Trying ~ to ~ typeset ~ #1}
  \bool_set_true:N \l_tmpa_bool
  \str_map_inline:nn {#1} {
    \__SIL_primitive_font_glyph_if_exists:nTF {##1} {}{
      \bool_set_false:N \l_tmpa_bool
      \typeout{Cannot ~ typeset ~ ##1}
      \str_map_break:
    }
  }
  \bool_if:nTF \l_tmpa_bool {\prg_return_true:} {\prg_return_false:}
}

\cs_generate_variant:Nn \__SIL_can_typeset:nTF {x}
\NewDocumentCommand\CanTypesetTF { m m m }{
  \__SIL_can_typeset:xTF{#1}{#2}{#3}
}
```

Figure 4: Checking whether a given string can be typeset with a given font

```
...
Arimo-Bold.ttf
Arimo-BoldItalic.ttf
Arimo-Italic.ttf
Arimo-Regular.ttf
Arsenal-Bold.otf
Arsenal-Bold.otf hist
Arsenal-Bold.otf smcp
Arsenal-Bold.otf ss01
Arsenal-Bold.otf ss02
Arsenal-Bold.otf swsh
...
```

Figure 5: Fragment of the font list

for this remark). Still, I found out that the engines choke when the number of fonts in the document exceeds 5500. I did not want to recompile the engines, so I employed several mitigation strategies:

1. The package does not load different sizes of a font to fit a spine. Instead, it changes the sizes of the rectangle that represents the spine, and then uses `\resizebox`. Generally, such resizing of fonts is a bad typographic practice; this is one of the rare cases when it seems to be appropriate.
2. The actual algorithm for choosing a random font has two stages. On the first stage we randomly select a font from the general list *and* save its number in the stack of opened fonts. When the size of this stack exceeds the limit, we no longer

use the general list, but randomly select the font from the stack.

With these changes the package was able to typeset Figure 2.

The code is now available at the Github repository github.com/borisveytsman/bookshelf. Peter kindly allowed me to take over the maintenance of the package on CTAN, so the new version with all these changes will be released after some code cleaning. There are some features I'd like to add, including colorblind palettes, streamlining the typesetting, making the package aware of the size of the actual book (so large books have larger spines).

It is difficult to find a “practical” application for this package. Still, it brought much fun to me. I am grateful to Peter for inventing it, and hope my extensions are welcomed by other users.

References

- [1] P. Flynn. *The bookshelf package*, 2020.
ctan.org/pkg/bookshelf
- [2] K. Goyal. *calibre User Manual*, 2024.
manual.calibre-ebook.com

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

Holon programming regained

Mitchell Gerrard

Abstract

One of the main inspirations for literate programming was a technical report entitled *Holon Programming: A Survey*, by Pierre-Arnoul de Marneffe. It was privately circulated among computer scientists in 1973. The document thereafter became a Borgesian mythical book, existing only in citations by Knuth. This article narrates the search-and-rescue mission of this rare book, and highlights a few of its innovations. The full report is available at github.com/holon-scribe/holon-programming.

1 Ancient history

And indeed, he composed a fair great book with figures, but it is not printed as yet that I know of.

— François Rabelais, *Pantagruel* (1532)

In 1973, a Belgian computer scientist named Pierre-Arnoul de Marneffe was finishing a report describing his ideal programming language. More on this later.

A few years prior, Edsger Dijkstra had circulated his *Notes on Structured Programming* [3]. These *Notes* marked a watershed in the computing community. Dijkstra urged the systematic use of now-commonplace programming constructs such as for loops, if/then/else statements, and subroutines. He also gave a method to write complex programs by starting with an abstract description and successively refining this description into smaller, more manageable chunks. But what should one call these chunks of related computation? Dijkstra called them “pearls”, regarding a program as a necklace strung from individual pearls; Donald Knuth wrote to Dijkstra: “We need another word for pearl, though; what should it be?” [7]

Prof. de Marneffe knew what the word should be. He had recently read *The Ghost in the Machine* by Arthur Koestler, in which Koestler coins the term “holon”, denoting the various “nodes on [a] hierachic tree which behave partly as wholes or wholly as parts, according to the way you look at them.” [12] The *holon* would be the unifying concept in de Marneffe’s synthesis of Dijkstra and Koestler. In December of 1973, de Marneffe privately circulated copies of his report entitled *Holon Programming: A Survey*.

2 Modern times

I try to reason, and I tell myself you’ll return.

— Roberta Flack, *Gone Away* (1970)

The year is now 2015. I had written my first few literate programs using Norman Ramsey’s `noweb`

tool [17], and was immediately smitten by this peculiar approach to programming. So I reread Knuth’s article that introduced literate programming, to seek the font of this love potion that was on my sleeping eyelids laid. And happy day—here were breadcrumbs: “The design of WEB was influenced primarily by the pioneering work of Pierre-Arnoul de Marneffe, whose research on what he called ‘Holon Programming’ has not received the attention it deserves.” [8] This statement was accompanied by two citations, one of them a 135-page report [6]. Yet when I searched for this report online, there were no books published under this name, there were no PDF scans, there was almost nothing save for a few tantalizing passages and descriptions of this mysterious document. Most strange.

Was *Holon Programming* a fictitious entry in *The Catalog of Lost Books* [20]? But there were extracts, and de Marneffe was a real author... surely this influential report hadn’t been lost to posterity.

I wrote to Prof. de Marneffe. He replied that he indeed had a copy in his files that he would scan and send to me, but he was recovering from a long hospital stay, so would do so after feeling better. Some time passed and I did not want to trouble Prof. de Marneffe further. I then wrote to Prof. Knuth. He replied: “I think I donated my copy to Stanford’s tech reports collection, but they don’t seem to have it”, and directed me to the only known library copy that was supposedly held in Germany. Knuth also enclosed a copy of the letter that he wrote to de Marneffe in 1974. This letter contained such specific references to the report that I was almost convinced *Holon Programming* was not a fabrication. Unfortunately some doubts remained, as the letter was dated April 1st.

The full letter is reproduced below (with permission).

3 Knuth’s letter

April 1, 1974

Prof. Pierre-Arnoul de Marneffe
Université de Liège
Service d’Informatique
Avenue des Tilleuls 59
B-4000 Liège, Belgium

Dear Prof. de Marneffe:

Thank you very much for sending me your survey of Holon Programming. I especially enjoyed your references to the non-computer literature (Koestler, Bernard-Shaw, Shanley, Mount Vernon, etc.) since computer scientists need to avoid insularity.

I believe you are making important strides toward the development of a new programming language. There still remain some unclear areas but you are obviously addressing the correct issues; the next thing to do (it seems to me) is to program several hundred examples!

For related reading I would suggest that you carefully study Ole-Johan Dahl's papers on SIMULA since his class concept is so close to the holon concept. Also I have just heard that Brian Randell of Newcastle has been working on a so-called PEARL system.

I found your report could have been improved if you had worked entirely with tree structures instead of converting to binary trees. The original tree structure is what is really relevant, and the Dewey notation for such structure is more directly suited to the operations you discuss. The binary tree is only a machine-oriented representation of the basic concept, the discussion should stay at a higher level.

Secondly, I found the report too preoccupied with details of implementation. The people by whom it is most important that this report be read are either able to visualize easily how to implement this sort of system, or else they are people who are not likely to care how it's implemented as long as it's handled sensibly. The important thing to stress is rather the conceptual issues of how holon programming differs from and improves on today's languages.

The example didn't come until page 100, while I expect most readers would have preferred to see it immediately. Since the program is almost self-explanatory, you can let it explain the language at the same time (integration of functions!).

Ideally there should be more examples of course.

The one example raises some interesting issues since the individual holons don't quite state their assumptions. In the very first holon, for example, it is not at all clear why you 'find first word starting character' instead of going right into 'find a word **etc.**' You must already have made a decision (a) that you wouldn't assume the text begins with a nonblank, (b) that there is going to be at this level an element of data representing the last-read character, (c) that the 'find a word' routine will already have its first character in hand, and (d) that there is no need to test for a message that has no words (only a full stop). As I recall when I was solving that problem, it took me a good five minutes to reach these decisions, during which I must have considered lots of alternatives. Once this step was made the rest of the program flowed naturally. My questions are: Where should we state these assumptions? Shouldn't we mention the existence of data representing the

last-read character, even though we don't want to specify its detailed structure until later?

These issues seem to arise repeatedly and I haven't a first conclusion about what we ought to do. That's why I suggest working out hundreds of examples, as being the best kind of eating to prove the pudding at this stage. On the other hand creating the holon implementation itself is equivalent to working out quite a few examples.

Thanks again for showing me your stimulating work. I myself must get on with the writing of volume 4 of my series, so I have little energy to devote to the development of languages, but I will do my best to see that other people working in the area are kept informed of what you are doing.

Sincerely,

Donald E. Knuth
Professor

P.S. Is there a place in Belgium whose postal code is B-6700 like the Burroughs computer?

4 The survey itself

Our Perdita is found.

—Shakespeare, *A Winter's Tale* (1611)

More time passed, revealing more false bottoms, but, eventually, kind librarians on both sides of the Atlantic arranged for that most elusive document to be sent from Hanover to Nebraska.

And so: After more than 50 years in hiding, Pierre-Arnoul de Marneffe's *Holon Programming: A Survey*, prefaced with Knuth's letter, is returned. Make its acquaintance at the address below.

 github.com/holon-scribe/holon-programming

I second Knuth's suggestion to jump straight to the `<Program Example>` section to get a feel for what it's all about. And then, in the hypertext spirit of *Hopscotch* [2], jump around and **go to** whichever chapter titles most draw you.

In this technical report filled with out-of-the-way observations, projected language features and imagined ecosystems (a full "holon operating system")—where is the literate programming? Well, if you attire the program example's bare pseudocode phrases with a `<` and `>` on either side, the holons transform into the code sections of **WEB**. So let's do just that: we'll take an extract of de Marneffe's program and translate its "holons" into corresponding sections of a **WEB** program. This program solves a problem from section 16 of Dijkstra's Notes; its details aren't relevant here. Two notes on the syntax: the **etc** keyword abbreviates unambiguous prefixes, and '`#`' followed by '`##`' brackets low-level statements.

5 A holon program and its WEB twin

odd inversion program

```
begin find first word starting character;
    repeat find a word and print correctly;
        until end of useful file
    end
```

find first word **etc**

```
begin read first symbol;
    while last read symbol is a space;
        do read next symbol
    end
```

:

read first symbol

```
begin declare lrs: character at
    odd inversion program level;
    # lrs ← RNC ##;
end
```

:

And here are the equivalent extracts written in WEB. The ⟨ Global variables ⟩ section approximates how, in de Marneffe’s language, one “declares a variable and specifies the scope by naming an outer holon.” [5]

1. This program is one possible solution to the problem posed in section 16 of Dijkstra’s Notes.

program *odd_inversion*;

```
var ⟨ Global variables 4 ⟩
begin ⟨ Find first word starting character 2 ⟩;
    repeat ⟨ Find a word and print correctly 3 ⟩;
        until ⟨ End of useful file 13 ⟩
    end.
```

2. ⟨ Find first word … 2 ⟩ ≡

```
begin ⟨ Read first symbol 11 ⟩;
    while ⟨ Last read symbol is a space 12 ⟩;
        do ⟨ Read next symbol 6 ⟩
    end
```

This code is used in section 1.

:

10. ⟨ Global variables 4 ⟩ +≡

trs: char; { last-read symbol }

This code is used in section 1.

11. ⟨ Read first symbol 11 ⟩ ≡

```
begin trs ← RNC; { read next character }
end
```

This code is used in section 2.

:

RNC: procedure, §15

The resemblance is uncanny. What de Marneffe did was show how a program can be written “in the order of its design”, using phrases mostly in natural language, in digestible sections of no greater than eight lines, that can be automatically “disentangled” (de Marneffe’s word) into a fully executable program. Knuth describes de Marneffe’s approach as “a way of taking a complicated program and breaking it into small parts. Then, to understand the complicated whole, what you needed is just to understand the small parts, and to understand the relationship between each part and its neighbors.” [10]

I won’t go on to give a book report of *Holon Programming*. Instead, I’ll assign further reading and then highlight a few more prefigurings of the literate programming we know today.

To rough in more of the context in which de Marneffe’s survey is, holon-like, embedded, I recommend reading Chapters 2, 3 and 5 of *The Ghost in the Machine* through the lens of Dijkstra’s Notes. Koestler’s arguments employ the language of computer science (via Herbert A. Simon); the metaphors and exact phrasings he uses to describe carrying out tasks closely echo those Dijkstra uses to describe fleshing out programs. They also share a fondness for pugilistic asides. In a happy coincidence, the section that de Marneffe singles out in Koestler, “How to Build a Nest”, contains three instances of the word “web” and four instances of “weaving”.

Now for the highlights.

We come across a few false friends in comparing de Marneffe’s language with Knuth’s. The **append** command refers to defining a new section, differing from the \equiv append operation in WEB. There is an appearance of suggested macro use; but unlike Knuth’s more straightforward macros, de Marneffe’s were to parameterize holons themselves, making them more procedure-like.

Other constructs are remarkable prototypes of those we know: the **text** command defines a section of prose to “explain the reason of some design decisions”; the **change** command is like Knuth’s change file mechanism, except the entire section must be replaced; the **etc** abbreviation keyword has turned into the ‘…’ shorthand in a WEB source file; the **output** command prints the holon program as intended for human eyes to an output device, somewhat akin to weaving; and the **synthesize** command outputs the program intended for machine consumption, akin to tangling.

Most wonderful!

But hol’ on there... if all these elements were present in de Marneffe, what exactly were Knuth’s contributions?

6 Woven WEBs

Nothing about WEB is really new; I have simply combined a bunch of ideas that have been in the air for a long time.

— Donald Knuth, *Literate Programming* (1984)

Let no one say that I have said nothing new; the arrangement of the subject is new. When we play tennis, we both play with the same ball, but one of us places it better.

— Blaise Pascal, *Pensées* (1657?)

Knuth's selection, rearrangement, and improvement upon ideas "in the air" was decisive. He saw through many of the irrelevant technical details in de Marneffe's report and grasped the essence. Knuth made the subtle but crucial design decision to bring the informal prose explanations to the forefront. (The `text` command in de Marneffe never appears in examples, and seems to be regarded as a simple "comment" mechanism, despite my hyping it just now.)

The importance of colorful language, metaphors, and rephrasings cannot be understated when thinking about the unreasonable effectiveness of literate programming. "We retain only what has been dramatised by language; any other judgment is fleeting." [1] The storytelling elements of a good literate program act as strong fixatives in our memory. And Knuth does not limit the prose to the "informal" portions; it spills over into the formal (code) portions as well. In all of Knuth's published literate programs, he follows most macro definitions and variable declarations with some explanatory comment.

As Knuth brings informal prose to center stage, he also casts the prettyprinted code to be its costar. Whereas de Marneffe banishes the lowly code statements to hide within ugly '#' and '##' curtains and leaves the holon names unmarred, Knuth reverses this: he lets the (formatted) code stand on its own and brackets the section names with (less-obtrusive) delimiters. Knuth does not at all want to give short shrift to the code. The typeset Pascal in WEB's woven output invites the reader to see how the code syncs up with its informal explanation above.

Knuth developed the first implementation of literate programming, with the `TEX` and `METAFONT` projects being the "proof in the pudding". WEB includes a host of features unforeseen by de Marneffe, such as commands to produce camera-ready programs. Alongside these projects Knuth brought the history of literature, typography, and book design to bear on this style of programming. Beautiful fonts, typeset code blocks, cross-referencing included with each section name, tables of contents, indices, mini-indices, appendices, bibliographies, figures . . .

But we are getting far afield. Briefly stated: there was a whole lot that was new in WEB.

7 Old yarns

*What threads were those, oh, ye Weird Ones,
that ye wove in the years foregone.*

— Herman Melville, *Pierre; or, The Ambiguities* (1852)

I've focused the discussion of the genesis of holon programming on the two authors de Marneffe cites the most: Dijkstra and Koestler ("The author really doesn't know to what extent the reader can grasp the holon concept without reading Koestler's book." [5]). But there was one other primary influence on de Marneffe's language. Who? Lo and behold: Knuth.

The program example used by de Marneffe to illustrate his holon language is predicated by two alternate program solutions to the same problem, given by Knuth in his letters to Dijkstra and Dahl [7]. We know de Marneffe was familiar with these letters because he cites and comments upon them. In Knuth's first program solution, we see the program only in its final, (mostly) executable, stage, but he does something remarkably close to de Marneffe. That is, Knuth composes a program out of small code sections not exceeding eight lines, each labeled with a natural language descriptor, constructed in "the order in which the decisions were made" [7], and systematically expanded and interleaved into a machine-readable form. The code sections, called "pearls" in the letter, are identified in the left margin of the "tangled" output. The second program solution is largely the same, but Knuth explicitly groups the small independent sections (here called "classes") in the top-down order of design. These programs differ in many details from de Marneffe's system, but the influence in language design clearly ran in both directions. And of course there are countless other contributors who helped bring about literate programming.

So I would like to express a desire to reprint some of the precursors in a slim anthology entitled *Pre-literate Programming*. This could include selections from de Marneffe; the PEARL system mentioned by Knuth in his letter to de Marneffe [18]; Dahl's SIMULA papers [4]; Naur's "Programming by action clusters" [14] and "Formalization in program development" [15]; Knuth on *Structured Programming* [7]; Babbage's "On a method of expressing by signs the action of machinery" [13]; excerpts from Dijkstra's Notes [3]; selections from Chapter 2 of Wilkes et al.'s book [22]; a rifacimento of Weinberg's book into a collection of aphorisms [21]; Towster's work [19];

portions of Jim Dunlap’s early compiler code with forty-character-long identifiers; a two-page spread exhibiting the impact typography has on program comprehension, with a non-typeset Algol program on one side and the same program typeset with executive editor for ACM Myrtle Kellington’s standards on the opposite side; Derek Oppen’s “Prettyprinting” [16]; and, as a specimen of good storytelling, something by Shirley Jackson. (We leave out the many shoots and buds of literate programming already gathered in Knuth’s early papers from [9] and “Computer Drawn Flowcharts” and pp. 229–235 in [11].)

Acknowledgments

Thanks Don Knuth for help in locating this report and for letting the 1974 letter be reproduced. Thank you librarians. Thanks Karl Berry, Udo Wermuth, and Andreas Scherer for your feedback and suggestions. And thank you Pierre-Arnoul Frédéric Guy Donat de Marneffe (1946–2023) for creating and sharing this groundbreaking work.

References

- [1] G. Bachelard. *The Dialectic of Duration*. Rowman & Littlefield, 2016.
- [2] J. Cortázar. *Rayuela (Hopscotch)*. Sudamericana, 1963.
- [3] O.J. Dahl, E.W. Dijkstra, C.A.R. Hoare. *Structured programming*. Academic Press Ltd., 1972. archive.org/details/Structured_Programming__Dahl_Dijkstra_Hoare
- [4] O.J. Dahl, K. Nygaard. SIMULA: an ALGOL-based simulation language. *Communications of the ACM*, 9(9):671–678, 1966.
- [5] P. de Marneffe, D. Ribbens. Holon programming. In *International Computing Symposium*, A. Günther et al., ed., Amsterdam, North Holland, 1973.
- [6] P.A. de Marneffe. *Holon Programming: A Survey*. Univ. de Liège, December 1973.
- [7] D.E. Knuth. A review of “Structured Programming”. Technical Report STAN-CS-73-371, Stanford Computer Science Department, Stanford University, Stanford, CA, 1973. i.stanford.edu/TR/CS-TR-73-371.html
- [8] D.E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.
- [9] D.E. Knuth. *Literate Programming*. CSLI, 1992.
- [10] D.E. Knuth. *Digital Typography*. CSLI, 1999.
- [11] D.E. Knuth. *Selected Papers on Computer Languages*. CSLI, 2003.
- [12] A. Koestler. *The Ghost in the Machine*. Macmillan, 1968.
- [13] P. Morrison, E. Morrison. *Charles Babbage and his calculating engines: Selected writings by Charles Babbage and others*. Dover, New York, 1961. archive.org/details/philtrans09445034
- [14] P. Naur. Programming by action clusters. *BIT Numerical Mathematics*, 9(3):250–258, 1969.
- [15] P. Naur. Formalization in program development. *BIT Numerical Mathematics*, 22(4):437–453, 1982.
- [16] D.C. Oppen. Prettyprinting. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2(4):465–483, 1980.
- [17] N. Ramsey. Literate programming simplified. *IEEE Software*, 11(5):97–105, 1994.
- [18] R.A. Snowdon. PEARN: an interactive system for the preparation and validation of structured programs. *SIGPLAN Notices*, 7(3):9–26, Mar. 1972.
- [19] E. Towster. A convention for explicit declaration of environments and top-down refinement of data. *IEEE Transactions on Software Engineering*, SE-5(4):374–386, July 1979.
- [20] T. Tuleja. *The Catalog of Lost Books: An annotated and seriously addled collection of great books that should have been written but never were*. Fawcett Columbine, 1989.
- [21] G.M. Weinberg. *The Psychology of Computer Programming*, vol. 29. Van Nostrand Reinhold New York, 1971.
- [22] M.V. Wilkes, D.J. Wheeler, S. Gill. *The Preparation of Programs for an Electronic Digital Computer: With special reference to the EDSAC and the use of a library of subroutines*. Addison-Wesley Press, 1951.

◇ Mitchell Gerrard
 mitchell dot gerrard (at) gmail
 dot com

On-demand production of TeX DVDs: first feedback

Jérémie Just

Abstract

Distributing TeX software on physical media has been a mission of the TeX user groups since their earliest days. As of 2024, with wide availability of high-speed permanent access to the Internet, the user groups agreed to switch the way this service is provided: the DVDs are now produced and shipped only on demand, by a group of volunteers spread over the world.

1 The historic TeX Collection DVD

Obtaining TeX software has been a major reason to join user groups, especially before the advent of widespread Internet access. The 1980s saw floppy disks being exchanged between user group members; with an increasing number of packages deposited on CTAN, distribution switched to CDs in the 1990s, and quickly to DVDs (Fig. 1).

As of 2023, the TeX Collection DVD contained more than 7.8 GB of TeX-related software, in three distributions (TeX Live, MiKTeX and MacTeX). In total, 3,300 copies were produced by DANTE for all TeX user groups, at a cost of 0.50 euro each. DANTE, TUG and a few other groups were still shipping it to all their members, but other groups, for example GUTenberg, were already shipping to members exclusively on-demand.

For TUG, the contribution to production, plus shipping of 750 DVDs, represented a budget item of \approx USD 3,120 in 2023.

2 Why stop mass-production?

The general feedback was that the vast majority of members didn't need or want a DVD, having sufficient bandwidth to access downloads. Some people were even asking not to get it, to reduce clutter or to better spend community money and time. Therefore making the DVD as a large bulk production and mailing didn't seem the best choice any longer.

Nevertheless, approximately 20 individuals ordered a DVD from the TUG online store in 2023, and I personally provided 6 copies to French users in the same year; thus availability on physical media still has a real use, in some situations.

So time had come to switch from producing thousands of DVDs of which only a handful would be used, to an on-demand production.

Motion 2023.3 was voted by the TUG board in August 2023, ratifying the discontinuation of

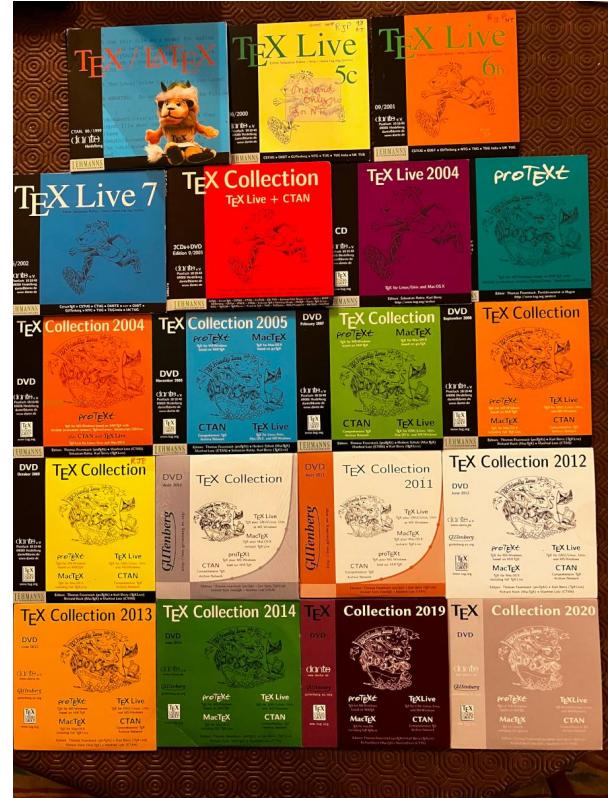


Figure 1: A collection of TeX Collection DVDs.
Courtesy Robert PAPOULAR.

TeX Collection DVD as a general TUG membership benefit.

3 A network of volunteers to burn DVDs on demand

A network of volunteers was initiated in spring 2024. The only requirement to join this network: have access to a DVD burner, with a few spare dual-layer DVDs at hand.

A web form hosted on TUG web server now allows to request DVDs.¹ In the past, a single disc (the “TeX Collection”) was able to store all four components (TeX Live, MiKTeX, MacTeX and a CTAN snapshot), but as of this year, this is no longer possible: the combined size exceeds the capacity of one disc, even dual-layer, so people have to select one or more of the four available DVDs.

New graphic designs for the DVDs and their sleeves were also conceived (Figs. 2 and 3) and will soon be made available to everyone.

The mailing list `texdvd@tug.org` can be used for discussion and special requests. Please do not post any personal information, such as your postal

¹ Thanks to Max Chernoff for developing this web form, along with its back-office interface, to handle DVD requests.

address, as the list archives are public (list archives and information: lists.tug.org/texdvd).

4 How can I get a TEX distribution, today?

First option: installing a TEX distribution over the internet is now far less tedious than fifteen years ago. If your computer has decent Internet access, give it a try!

tug.org/texlive/acquire-netinstall.html

Second option: the DVD images are all publicly available as ISO files. If a friend or relative nearby has a broadband internet access and a DVD burner, they can download them and burn a disc for you. The up-to-date links are given on this page:

tug.org/dvd/

Third option, the new one: rely on our network of volunteers! Just fill in and submit the DVD request form here:

tug.org/dvd/request.html

You will be contacted by one of the volunteers by private email. An arrangement will be made between you and the volunteer about the price of the DVD and shipment, and the way of transferring the money, and you will need to provide your preferred mailing address.

Please bear in mind that this service is being provided by people in their spare time, and that no profit is made on it. If you have the impression your request has got lost, or if you're having difficulties

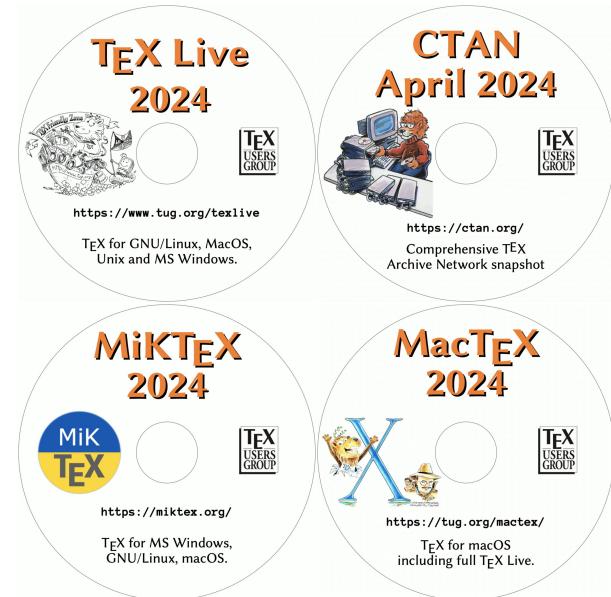


Figure 2: The four DVD designs, as of 2024...

contacting the person who has volunteered to handle your request, please send an email to the main mailing list for anything related to TEX DVDs:

texdvd@tug.org

At this writing, it's been about 4 months that we have been working this way. About 26 DVDs have been shipped, for prices ranging from 3.50 euros (1 DVD, domestic shipping) to 10 euros (4 DVDs, intercontinental shipping). Payments have mostly gone through PayPal or direct bank transfers.

I personally have greatly enjoyed each time I dropped a DVD in the nearest mailbox, knowing that it would soon arrive at the other end of the world, ready for a TEX installation!

5 How can I help?

If you have access to a DVD burner and are willing to give a hand distributing TEX software, please consider joining the group of volunteers! Just send a message to texdvd@tug.org.

Another way to contribute to TEX and the user groups that support it is donating at tug.org/donate, or to another group. All donations are welcome!

6 Acknowledgements

First of all, I would like to thank the hundreds of individuals who have been instrumental in developing the software that we're now helping to disseminate across the globe on those four DVDs. I'm also very grateful to those who have consistently packaged the distributions and made ISO images available, each year. Finally, I want to acknowledge the work of all people who volunteered for this new mission.

◊ Jérémie Just
Lyon, France
[jeremy\(at\)jejust.dotsfr](mailto:jeremy(at)jejust.dotsfr)
<https://tug.org/dvd>

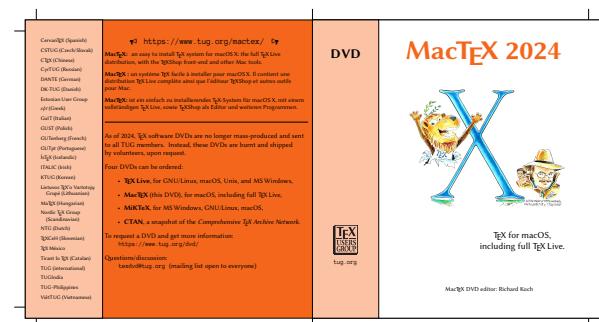


Figure 3: ... and one of the sleeves.

Profiling T_EX input files

Martin Ruckert

Abstract

A profiler is a tool used by programmers to analyze the runtime behavior of the code they write. The profiler can map the CPU time of a program to specific files and lines, or it can map the time to individual procedures. This information is necessary if a programmer wants to optimize the code for speed.

No such tool has been available to date to programmers who write macro packages for T_EX. This paper presents `texprof` and `texprofile`, two programs working together to profile T_EX input files.

1 Who needs a profiler?

The T_EX profiler is a tool for programmers writing T_EX macros. This does not mean that an author who occasionally writes a T_EX macro should use or even needs to use this tool. Optimizing a macro for speed should be done only if the macro is used *very often*. To get a feeling for what “*very often*” means, consider the following: Under reasonable assumptions (200 watt peak power consumption of your PC and 500g of CO₂ emission per kWh electric power use) one second of CPU time results in 28mg of CO₂ emission. Again under reasonable assumptions (2370g of CO₂ emission per liter fuel and 6l fuel consumption per km) driving 200km results in 28kg of CO₂ emission. That means that you need to save millions of seconds in CPU time before it has any substantial impact on your CO₂ footprint or your budget.

So for the occasional macro writer there are better opportunities to invest time and intelligence than optimizing macros for speed. But of course there are macro packages for T_EX that have millions of users that use these macros in multiple runs every day, and if you are the programmer of such a package, you might be interested to know if there are opportunities for optimization, where these opportunities are hiding in your code, and how much you might gain when optimizing this code. Maybe even more important, a profiler can tell you where not to look for optimizations, and—after the optimization—if the changes to your code had the desired effect. As a general rule, you should never optimize code for speed without using a profiler.

2 How does the T_EX profiler work?

2.1 Mapping commands to files and lines

Every T_EX engine is an interpreter that executes the built-in commands of T_EX, like creating a horizon-

tal box, incrementing a count register, or adding a character to the current paragraph. The T_EX profiler, called `texprof`, is such an engine with extensions to map every command to a file and a line in that file. If `texprof` reads such a command from an input file, it can determine the file name and the line number from the data structures that every T_EX engine maintains to display good error messages. If on the other hand, such a command was part of a format file, the file name and the line number is not known. We will see below how using a format file can be avoided.

But even if we avoid using a format file, many commands are not read directly from an input file, instead they come from expanding a macro. When a macro is defined, T_EX stores the commands that belong to the macro body in a hash table, and when the macro is used, T_EX retrieves the commands from the hash table and inserts them into the input. Since the file and line are known when a macro is defined, this information can be stored in the hash table and retrieved along with the commands when a macro is expanded. The same mechanism comes into play when T_EX reads ahead, for example when scanning a keyword, and discovers that the commands seen must be pushed back into the input for later processing: the commands take their file and line numbers with them.

There are a few rare cases where this mechanism does not work. For example T_EX inserts a pair of curly braces around an output routine to make sure that commands executed in an output routine do not have unexpected global effects. These extra commands are marked as coming from the “system” file in line zero.

There are other “line numbers” in the “system” pseudo-file that the profiler will use. T_EX invokes system procedures like breaking a paragraph into lines or writing a page to the DVI file that can be quite time consuming. It would be misleading if the time spent in these routines would be mapped to whatever command happened to be executed at the end of a paragraph or caused the page builder to eject a page. So `texprof` associates these times with the “system” pseudo-file and uses the line number to indicate the responsible procedure in T_EX.

2.2 Mapping execution times to commands

Most of T_EX’s commands, but not all, are executed in T_EX’s `main_control` procedure. There we find a loop that reads a command and then executes the command by branching to its code to execute it using the so-named `big_switch` (the label in the Pascal code). `texprof` looks up the current time at

the start of each iteration of this loop. The time is taken from a hardware clock using a low-level routine provided by the operating system. The time taken from the clock is the start time of a new time interval and simultaneously the end time of the previous time interval. After reading the start time, `texprof` continues with the normal processing of `TEX` and reads the next command from its input stack. Once the command is known, `texprof` takes note of the command, its file, its line (and the macro it comes from—but let's focus for the moment on commands; macros will be explained later). Then normal processing continues and the command is executed using the `big_switch` which ends with a jump to the beginning of the loop. There `texprof` will again look up the time, compute the time difference and record command, file, line, and time difference in a large array.

Occasionally, a command is “reswitched”. That means, it is replaced in the `big_switch` by another command and the `big_switch` is used again to execute it. `texprof` ignores this replacement and will record the entire time together with the command, file, and line that was obtained at the beginning of the current iteration. This introduces some imprecision into the measurements but does not cause significant errors.

A much bigger effect on the association of time to file and line is caused by the existence of `TEX`'s `main_loop`. The `big_switch` will jump to this loop whenever it encounters a character and it will stay in this loop as long as the commands are the typical commands found in plain text: characters, spaces, kerns, ligatures, font changes, and a few more such things. The complete time spent in this loop is then recorded with the command, file, and line that started the loop. So the time used to process an entire paragraph might be associated with the first letter of that paragraph. The decision to forgo a more precise attribution of time in this case is justified by the following considerations: First, such a paragraph is normally processed only once and the time it takes is usually not a significant fraction of the total run time. Second, using a profiler, we are usually not interested in the time spent on letters, spaces and other parts of plain text. After all, no author optimizes the text for the speed of processing it. And finally, this reduces considerably the number of time intervals that `texprof` needs to record.

`TEX`'s `main_control` procedure exits when executing the “stop” command. This ends the recording of time intervals. As part of `TEX`'s closing procedure, `texprof` will write all the data collected to a binary file. The name of this file is obtained by

appending `.tprof` to the `\jobname`. All further processing of the collected data is not done by `texprof` but by a second program called `texprofile`. Its use will be illustrated below.

2.3 The problem of measuring time

Operating systems usually provide several clocks to choose from. `texprof` uses the common function `clock_gettime` to measure the CPU time of the current thread in nanoseconds. A measurement of nanoseconds seems like very precise information, but the actual precision is more on the order of a hundred microseconds, for several reasons. First, modern computers run many different processes at the same time and switch the available CPUs from one process to another. The time to switch processes includes of course the time needed to swap out and swap in the contents of memory caches. So when a process was recently swapped in, a load instruction from a location in main memory might take considerably longer if that memory location is no longer in the cache. This extra time is then associated with the `TEX` command that happens to be executing.

Second, modern CPUs use a technique called frequency scaling in order to reduce energy consumption. While you just type text in the editor, you laptop might run with a frequency below 1 GHz; shortly after you have started `TEX`, your operating system might notice that there is a lot of work to do and increase its clock frequency to 4 GHz. All commands that are executed after this change will need less time than the same commands before this change.

Third, in recent years manufacturers have begun to combine on one chip a few performance cores with a few efficiency cores. The former use sophisticated superscalar pipelines to execute multiple instructions per clock cycle; the latter use simple in-order execution which requires far less power and produces far less heat but might need multiple clock cycles per instruction. So if your operating system decides to move `texprof` from an efficiency core to a performance core in the middle of running, this has a drastic effect on the command times. The operating system might move it even back to the efficiency core if `texprof` starts to do file input/output which causes it to wait for the disk.

There are a few possibilities to mitigate these effects like computing the average over multiple runs or using synthetic times, but none of them is currently implemented.

2.4 Mapping computing times to macros

When `TEX` encounters an active character or a control sequence, it knows it has to execute a macro. It

looks up the list of commands that form the body of the macro and pushes this list on its input stack. When \TeX needs the next command from the input, it takes it from the topmost list on the input stack. And when \TeX reaches the end of the topmost list, it pops it from the stack and continues to read commands from where it was before in the next lower list on the stack. The input stack is used not only for macro calls, but also for implicit calls to other routines. The page builder will, for example, push the output routine on the input stack when a new page is ready. Or at the beginning of a paragraph, the system will push the commands specified with `\everypar` on the stack. Entire files are pushed on the stack when you use `\input`. It is also very common that \TeX reads ahead, for example to check for a possible keyword, and pushes unused tokens back on the input stack to be read again if the keyword was not found.

Most of the information `texprof` needs to keep track of macros can be found on \TeX 's input stack—but not all of it. Notably, the input stack will not contain information about the correct nesting level of the macros. The reason for this is a clever optimization that \TeX uses on its input stack called last call optimization: Before a new macro body is pushed on the input stack, \TeX checks repeatedly if the topmost list on the input stack is already empty. And if so, it will remove the empty list from the stack. Only after all empty lists have been removed the body of the new macro is pushed. Using this optimization, a loop that is implemented by a recursive macro, calling itself as the last action of the macro body, can run without overflowing the stack. Unfortunately this technique will remove a macro from the input stack while its last sub-macro is still running; and it will push new macros at a lower nesting level than their “true” nesting level.

So `texprof` adds information about the “true” current macro nesting level to \TeX 's input stack and maintains a separate stack that contains information about all macros up to the true nesting level: the name as well as the file and line number of the macros definition. This stack provides information about the true begin and end of a macro call which is recorded together with the timing information of executed commands.

3 Analyzing profiling data

The raw data that `texprof` writes to the output file is just a long list of thousands of “command file line time” records interspersed with records that reflect the changing of the macro stack. Extracting

useful information from this data is the job of the `texprofile` program.

To explain the use of `texprof` and `texprofile`, it is best to use examples. For the first example, I was looking for a large document with a focus on text. Searching the internet for such an example, I found a \TeX version of the bible (github.com/vermiculus/bible). With a few changes I made it use the “plain” \TeX format so that it makes only a limited use of macros. Most macros are taken from plain \TeX , but there are also some user defined macros. Running `texprof -prof bible` will create `bible.tprof` with a size of 17 Mbyte. Running `tprof bible` without further command line options will print the following summary:

Total time measured:	728.92 ms
Total number of samples:	2157642
Average time per sample:	337.00 ns
Total number of files:	69
Total number of macros:	1097
Maximum stack nesting depth:	7

You can use command line options to specify which data tables `texprofile` should display and how it should display the information.

3.1 The top ten lines

Given the `-T` option, for example, `texprofile` will traverse the data, add up the times for each file and line combination separately, then sort the results, and display the ten lines with the highest cumulative times: the “Top Ten” lines. The output is shown in Fig. 1.

The first line in the table is attributed to the “system” pseudo-file. The entry shows the accumulated time for an important system routine using the line number to identify the specific routine like producing the output DVI file (`shipout`), or a bit further down breaking a paragraph into lines (`linebrk`), or breaking the document into pages (`buildpg`). These times do not depend on the use of macros but simply on the size of the document.

From the next line, we can see that line 29 of `bible.tex` is responsible for 17.63% of the total run time and therefore is a good candidate for optimization, which we will try to do in the next section. The line by itself is quite fast, on average only $2.85\ \mu\text{s}$ are spent on this line, but the line is used very often: 54649 times.

The fact that the remaining six lines all contribute less than 1% to the overall runtime means that we need not consider any of them for optimization.

file	line	percent	absolute	count	average	file
system	shipout	17.68%	156.05 ms	1130	138.09 μ s	system
5	29	17.63%	155.65 ms	54649	2.85 μ s	bible.tex
system	linebrk	15.21%	134.26 ms	25777	5.21 μ s	system
system	buildpg	1.69%	14.89 ms	55190	269.00 ns	system
5	56	0.86%	7.61 ms	4750	1.60 μ s	bible.tex
5	15	0.62%	5.43 ms	6183	878.00 ns	bible.tex
3	555	0.47%	4.17 ms	8549	487.00 ns	plain.tex
3	1204	0.28%	2.44 ms	3390	719.00 ns	plain.tex
3	1201	0.26%	2.33 ms	2260	1.03 μ s	plain.tex
3	1203	0.25%	2.20 ms	2258	973.00 ns	plain.tex

Figure 1: Running `tprof -T bible`

file	line	percent	absolute	count	average	file
system	shipout	18.35%	156.29 ms	1130	138.31 μ s	system
system	linebrk	15.64%	133.23 ms	25777	5.17 μ s	system
5	29	12.85%	109.48 ms	60839	1.80 μ s	bible-opt.tex
3	666	1.95%	16.61 ms	55847	297.00 ns	plain.tex
system	buildpg	1.74%	14.85 ms	55190	269.00 ns	system
5	55	0.78%	6.67 ms	3552	1.88 μ s	bible-opt.tex
5	15	0.63%	5.39 ms	6183	871.00 ns	bible-opt.tex
3	555	0.49%	4.18 ms	8549	489.00 ns	plain.tex
3	1204	0.29%	2.43 ms	3390	716.00 ns	plain.tex
3	1201	0.27%	2.29 ms	2260	1.01 μ s	plain.tex

Figure 2: Running `tprof -T bible-opt`

3.2 Optimizing a macro

Line 29 of `bible.tex` defines the `\Verse` macro (formatted on two lines here for *TUGboat*):

```
\def\Verse{\global\advance\vcount
by 1${}^f\the\vcount$}
```

It is used to add the number of each verse, in small print and raised a bit above the baseline, to the beginning of every verse, like this:

17 And Moses and
the congregation to
families, by the ho
upward, by their pe
19 As the LORD

We optimize this macro for speed: The `\global` prefix is not needed because the macro is used only on the global level. `by` is an optional keyword and can be left out. Any literal constant like “1” is stored in the macro body as a sequence of characters which is rescanned and converted to an integer every time the macro is called. It is more efficient to use one of TeX’s registers instead. Last but not least, using math mode just to raise a number and

use a small font is a lot of processing for a simple effect. Here is the optimized version:

```
\newcount\1 \1=1 \newdimen\3 \3=3.6pt
\def\Verse{\advance\vcount\1
\leavevmode\raise\3
\vbox{\sevenrm\the\vcount}}
```

It uses two registers for the necessary constants and requires a call to `\leavevmode` because `\raise` is not allowed in vertical mode.

The top ten lines after optimization are shown in Fig. 2. Line 29 of `bible.tex` dropped from second place with 17.63% down to third place with only 12.85%. But this is not the full story: New is line 666 of `plain.tex` on fourth place with a 1.95%. So we get an overall speed-up of almost 3% from 17.63% down to 14.80%.

If we want to know what caused the increased use of plain TeX, looking at the call graph can shed some light on it.

3.3 The call graph

The call graph gives us information on a higher level of abstraction than what we gain from looking at the top ten lines. Consider if a different layout distributed the macro in line 29 that we have considered over 10 lines. We would have had 10 entries with about 1.8% each and none of them would have

	time	loop	percent	count/total	macro
\Verse					
	174.51 ms		24.64%	*	\Verse
	101.50 ms		58.16%	31011	\Verse
	73.01 ms		41.84%	31011/31011	\leavevmode
\output					
	121.22 ms		17.11%	*	\output
	1.15 ms		0.95%	1130	\output
	120.07 ms		99.05%	1130/1130	\plainoutput
\plainoutput					
	120.07 ms		16.95%	*	\plainoutput
	106.71 ms		88.87%	1130	\plainoutput
	6.99 ms		5.82%	1130/1130	\makeheadline
	2.76 ms		2.30%	1129/1130	\pagebody
	2.75 ms		2.29%	1130/1130	\makefootline
	859.36 μ s		0.72%	1130/1130	\advancepageno
\leavevmode					
	73.01 ms		10.31%	*	\leavevmode
	20.09 ms		27.52%	31011	\leavevmode
	52.92 ms		72.48%	495/1130	\output

Figure 3: Running `tprof -G bible-opt`

made it to the top of the list. The time recorded for a macro, on the other hand, does not depend on the layout of your source files. A macro gives a sequence of commands a common name, typically expressing its purpose, or the task that it will accomplish. To accomplish a task, a macro usually calls other macros, that we call child macros in the following. Such a child macro in turn might call again its own child macros. Along the chain of macro calls, a macro might eventually even call itself creating a recursive loop (as we will see below) where a macro becomes its own ancestor.

So when we look at the runtime of `TeX` from the macro perspective, we want to know how much time was spent in a certain macro, including all its descendants, because this is the time used to accomplish the task that the macro name promises to accomplish. We call this the cumulative time for the macro. Further we want to know how the cumulative time splits up into the time used by the macro itself and the time used by each of its child macros. This is the information that we gain by looking at the call graph. Fig. 3 shows the four macros that take the greatest percentage of the total run time.

We see the `\Verse` macro on top; 174.51ms are spent executing this macro which is almost a quarter of the total run time. But during the 31011 calls to this macro only 101.50ms or 58.16% of the 174.51ms are spent on the `\Verse` macro itself while the remaining 73.01ms are spent on calls to `\leavevmode`.

Looking at the last group of entries in Fig. 3, we see how the time used for `\leavevmode` is spent: roughly one quarter is spent on `\leavevmode` itself while the remaining three quarters are due to 495 calls to the `\output` routine. The total number of calls to the `\output` routine is 1130, which equals the number of pages of the document.

From the remaining entries, one for `\output` and one for `\plainoutput`, we see that `\output` does little more than call `\plainoutput`, which does most of the work itself, delegating only a small fraction of the work to properly named child macros.

3.4 Emulating `pdftex`

Our second example is `texprof` itself; to be precise: its documentation. `texprof` is an extension of `TeX`, and since `TeX` is implemented by a “literate” program, `texprof` is implemented by extending it. This literate program can be processed to obtain `texprof.c` from which a compiler can create an executable. Further it can be processed to obtain `texprof.tex` from which `tex` or `pdftex` can create a nicely typeset document.

Surprisingly creating a PDF with `pdftex` is significantly slower (2327ms) than creating a DVI file with `tex` (273ms). Of course, PDF is a much more complex file format than DVI and this accounts for some of the differences, but even if the creation of the PDF output is disabled (1602ms), there remains a considerable time difference. Just running

file	line	percent	absolute	count	average	file
9	156	20.29%	522.50 ms	225137	2.32 μ s	cwebacromac.tex
9	157	12.86%	331.08 ms	140088	2.36 μ s	cwebacromac.tex
9	158	9.13%	235.03 ms	140938	1.67 μ s	cwebacromac.tex
9	159	5.88%	151.27 ms	115319	1.31 μ s	cwebacromac.tex
9	172	3.68%	94.64 ms	15759	6.00 μ s	cwebacromac.tex
9	173	3.18%	81.95 ms	36954	2.22 μ s	cwebacromac.tex
system	shipout	3.16%	81.27 ms	775	104.87 μ s	system
system	linebrk	3.14%	80.93 ms	27370	2.96 μ s	system
9	152	2.16%	55.61 ms	67026	829.00 ns	cwebacromac.tex
9	166	1.86%	47.95 ms	13042	3.68 μ s	cwebacromac.tex

Figure 4: Running `tprof -T texprof`

```

156 \def\addtokens#1#2{\edef\addtoks{\noexpand#1={\the#1#2}}\addtoks}
157 \def\poptoks#1#2|ENDTOKS|{\let\first=#1\toksD={#1}%
158   \ifcat\noexpand\first0\countB=\#1\else\countB=0\fi\toksA={#2}%
159 \def\maketoks{\expandafter\poptoks\the\toksA|ENDTOKS|%
160   \ifnum\countB>`9 \countB=0 \fi
161   \ifnum\countB<`0
162     \ifnum0=\countC\else\makernote\fi
163     \ifx\first.\let\next=\maketoksdone\else
164       \let\next=\maketoks
165       \addtokens\toksB{\the\toksD}
166       \ifx\first,\addtokens\toksB{\space}\fi
167     \fi
168   \else \addtokens\toksC{\the\toksD}\global\countC=1\let\next=\maketoks
169   \fi
170   \next
171 }

```

Figure 5: Lines 156 to 171 of `cwebacromac.tex`

`texprof -prof texprof` will not suffice to find the source of the slowdown because `texprof` produces a DVI file and runs, when considering the profiling overhead, at approximately the same speed (443ms) as `tex`. The difference in speed is obviously caused by macros that are used only when producing PDF output. So we have to make `texprof` pretend to be `pdftex`. This can be achieved by processing a few macro definitions as shown in Fig. 7 before processing `texprof.tex`.

Using the file `fakepdf.tex` with these definitions, running `texprof -prof -jobname=texprof \input fakepdf.tex \input texprof.tex` will take 1771ms as expected. The top ten lines are shown in Fig. 4; they reveal that almost half of the runtime is caused by only four lines, 156–159, in file `cwebacromac.tex`. These lines, shown in Fig. 5, define macros `\addtoks`, `\poptoks`, and `\maketoks`. For information on the purpose of these lines, we can consult the call graph. Fig. 6 shows the three macros that take the largest percentage of the runtime. Let's consider them one by one.

3.5 Recursive macros

On top is the macro `\pdfnote` followed by the file number 4 and line number 152 in square brackets. This extra information is produced when using the `-i` option of `texprofile` and is needed to distinguish two macros that happen to have the same name, as we will see below. The macro `\pdfnote` creates links to the different sections of the documentation. `\pdfnote` is called 8473 times and each call makes a call to `\maketoks` which is responsible for almost all time needed for `\pdfnote`.

Each call to `\maketoks` in turn delegates most of its work to `\next`. If we look at the file and line information of `\maketoks` and `\next`, we discover that both macros are defined in the same file and on the same line 159. In Fig. 5, we see in line 159 the definition of `\maketoks` and in line 164 a `\let` command that makes `\next` an alias for `\maketoks`. The call to `\next` in line 170 ends the definition of `\maketoks`. So in fact `\maketoks` calls itself recursively. A recursive macro like this where the recursive call is at the very end of the macro is called “tail

	time	loop	percent	count/total	macro
\pdfnote [7,152]					
1.30 s			61.17%	*	\pdfnote [7,152]
26.54 ms			2.04%	8473	\pdfnote [7,152]
1.21 s			93.24%	8473/9271	\maketoks [7,159]
46.59 ms			3.58%	24230/28824	\pdflink [7,24]
14.67 ms			1.13%	4507/4507	\L [[5,334]
99.89 μ s			0.01%	80/80	\ETs [5,177]
54.34 μ s			0.00%	57/57	\ET [5,176]
404.00 ns			0.00%	1/3	\glob [4,166]
\maketoks [7,159]					
1.24 s			58.35%	*	\maketoks [7,159]
4.67 ms			0.38%	9271	\maketoks [7,159]
1.21 s			97.76%	9271/130811	\next [7,159]
14.59 ms			1.17%	9271/140082	\poptoks [7,157]
8.51 ms			0.69%	9271/225136	\addtokens [7,156]
\next [7,159]					
1.21 s			57.05%	*	\next [7,159]
53.94 ms			4.44%	130811	\next [7,159]
501.23 ms			41.29%	142326/225136	\addtokens [7,156]
462.00 ms			38.06%	130811/140082	\poptoks [7,157]
182.12 ms			15.00%	28737/28737	\makenote [7,172]
12.19 ms			1.00%	9271/9271	\next [7,174]
2.53 ms			0.21%	1456/2254	\makenote [7,154]
0.00 ns	1.19 s		0.00%	121540/130811	\next [7,159]

Figure 6: Running `tprof -G -i texprof`

recursive” and is optimized by TeX to run without growing the input stack as explained before.

The \next macro distributes the work among \addtokens and \poptoks and some calls to the \makenote macro. The 9271 calls of \next in the \maketoks macro eventually end in 9271 calls of \next as defined in line 174 where \next is redefined when the final “.” is found. Because \next calls itself, it is its own child macro and its own parent macro at the same time. This has consequences for the attribution of the cumulative times as shown in the call graph.

The first line in the table for the \next macro shows the total time spent in the next macro as 1.21 seconds; the following lines give a breakdown of these 1.21 seconds; the times given in their first column should add up to 1.21 seconds and the percentages given should add up to 100%. If `texprofile` only determined for each child macro the start and the end time and added up the time differences, the values for \next as a child macro of itself would come to 1.19 seconds, as shown in the column labeled “loop”. But when the \next child macro returns, all of that time is already included in the time shown in the previous lines. Therefore `texprofile`

maintains for each child two accumulators for the elapsed time: For the time shown in the column labeled “loop”, `texprofile` adds up the time differences observed at the return of a child macro. For the time shown in the column labeled “time”, it subtracts from the time differences observed at the return of a child macro all those time differences that were added to the macro itself or one of the other child macros since the start of the macro because these differences are already accounted for in the time breakdown. In a simple loop like the one we have here, all the time in \next as a child macro are already taken care of in \next as the parent macro. So the time column shows 0.00 nanoseconds. For more complex recursive loops this is not always the case.

4 Command line options and primitives

The command line options of `texprof` match those of other TeX engines. The only addition is the `-prof` option to switch profiling on right from the start. To profile only selected parts of a file, you can use the primitives \profileon and \profileoff.

The command line options of `texprofile` follow the general rule that options that select data tables use upper case letters and options that change the presentation of the data use lower case letters. We have seen already the `-T` option for the “Top Ten” table, the `-G` option for the call graph table, and the `-i` option to annotate (ambiguous) macro names with file and line numbers. `texprofile` can also display the cumulative times by input files with the `-F` option, by input lines with the `-L` option, or by `TeX` command with the `-C` option. Further the `-R` option displays the raw times for each and every command that was profiled. This table can get very large. It is useful if profiling was switched on for only a short time or if the data is sent to a file for further processing.

If the table data is intended for further processing, the `-m` option favors machine readability over human readability. Whereas by default times are displayed using an appropriate unit, either seconds `s`, milliseconds `ms`, microseconds `us`, or nanoseconds `ns`, the option `-m` will display all times in nanoseconds without specifying a unit.

The option `-pn` will suppress lines in the tables that fall below n percent. The option `-tn` with $1 \leq n \leq 100$ modifies the `-T` option to show the “Top n ” lines. The option `-s` modifies the `-R` option to include in the table information about the changes in the macro stack.

5 Improvements, workarounds, and future work

Current versions of `texprof` and `texprofile` are available from github.com/ruckertm/HINT. Since the first presentation of the `TeX` profiler at the TUG 2024 conference in Prague, a few improvements have

```
\def\pdftexversion{140}
\def\pdfoutput{1}
\def\pdfdest#1fith{}
\def\pdfendlink{}
\def\pdfannotlink#1{goto num#2%
  \Blue#3\Black\pdfendlink{#3}}
\def\pdfoutline goto#1 #2 #3{}
\def\pdfcatalog#1{}
```

Figure 7: The file `fakepdf.tex` implementing stubs for pdftEX primitives

been made. Most notably, the format files now contain file and line information for all source files used in generating the format. Therefore the attribution of runtime to an unknown file should now be a rare exception.

The method shown above to make `texprof` expand macros as `pdftex` would do is a workaround. As can be seen in Fig. 7, the stubs for the pdftEX primitives merely match the specific uses of these primitives in the given files. Some primitives, e.g. `\pdfannotlink`, have a complicated syntax that is easy to implement for engine primitives but quite complicated to achieve with `TeX` macros. Here some future work is necessary, either to make the implementation of macros with the desired syntax simple or to add a command line switch to `texprof` to make the necessary stubs available as engine primitives.

◊ Martin Ruckert
 Hochschule München
 Lothstrasse 64
 80336 München
 Germany
[martin.ruckert \(at\) hm dot edu](mailto:martin.ruckert(at)hm.edu)

Markdown themes in practice

Vít Starý Novotný

Abstract

The Markdown package for \TeX supports themes that allow \TeX nicians to tailor the presentation of Markdown and YAML content on the page. In this article, I will show the current state of Markdown themes using the example of \LaTeX templates that I developed for the International Software Testing Qualifications Board (ISTQB). Readers will leave with actionable steps to create or modify Markdown themes for \LaTeX , and insights into extending these principles to other \TeX engines.

Introduction

Although \TeX has beautiful output, its input macro language is an acquired taste for many authors. The Markdown package for \TeX allows authors to type familiar Markdown and YAML directly into a \TeX document and receive a similarly beautiful output.

In my previous article, I introduced Markdown themes [5]. Much like CSS stylesheets, Markdown themes allow \TeX nicians to tailor the presentation of Markdown and YAML content without complicating the document markup for authors. While that article used simple examples to explain the basic concepts behind Markdown themes, it did not demonstrate their application on a larger scale in real-world software projects.

In July 2023, I began working with the International Software Testing Qualifications Board (ISTQB) to help them typeset their certification study materials from Markdown and YAML sources. In this article, I discuss my work as a case study of using the Markdown package in a real-world software project.

Project overview

In my work, I developed a \LaTeX document class and six Markdown themes [1].

The \LaTeX document class is named `istqb` and it is stored in file `template/istqb.cls`. It implements the design of all ISTQB documents, defines the meaning of common Unicode characters, and defines \LaTeX markup such as `\istqbunnumberedsection`, `\istqblandscapebegin`, and `\istqblandscapeend`.

The Markdown themes are named `istqb/*` and stored in files `template/markdowntheme*.tex` and `*.sty`; see also Figure 1. Here is what they do:

- The theme `istqb/common` enables Markdown syntax extensions, implements the loading of YAML language definitions and document metadata into \TeX macros, and defines the mapping

between Markdown elements and \LaTeX markup. The remaining themes are based on this theme and they implement support for specific types of ISTQB documents.

- The `istqb/body-of-knowledge` and `syllabus` themes are used in ISTQB Body of Knowledge and Syllabus documents. At the time of writing, the themes implement no extra functionality.
- The theme `istqb/sample-exam` implements the loading of YAML question definitions into \TeX macros in ISTQB Sample Exam Questions and Answers documents. The following two themes are based on this theme.
- The theme `istqb/sample-exam/questions` implements the typesetting of questions in ISTQB Sample Exam Questions documents.
- The theme `istqb/sample-exam/answers` implements typesetting of answer keys and answers in ISTQB Sample Exam Answers documents.

In the rest of this article, I show the main concepts behind Markdown themes using the examples of ISTQB Sample Exam Questions and Answers documents, which use the themes `istqb/sample-exam/questions` and `/answers`.



With Markdown themes, your document can wear many different disguises, just like the wolf.

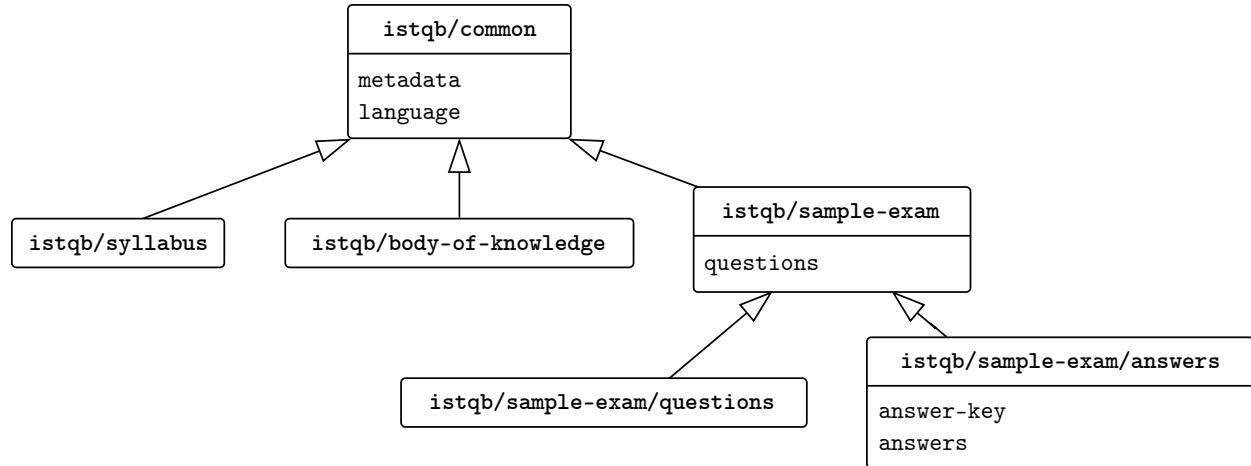


Figure 1: A class diagram of the six Markdown themes that I developed for the International Software Testing Qualifications Board (ISTQB). The snippets `metadata`, `language`, `questions`, `answer-key`, and `answers` specify the public interface of the themes and arrows specify inheritance.

Questions					
Question #1 (1 Point)					
What is the answer to life, the universe, and everything?					
<ul style="list-style-type: none"> a) 24 b) 42 c) 64 d) 84 					
Select ONE option.					
Question #5 (1 Point)					
What's France's capital?					
<ul style="list-style-type: none"> a) Berlin b) Madrid c) Paris d) Rome 					
Select ONE option.					
Question #6 (2 Points)					
Which two of the following animals are classified as mammals?					
<ul style="list-style-type: none"> a) Shark b) Dolphin c) Eagle d) Whale e) Crocodile 					
Select TWO options.					

Answer key																									
<table border="1"> <thead> <tr> <th>Question Number (#)</th> <th>Correct Answer</th> <th>Learning Objective (LO)</th> <th>K-Level</th> <th>Number of Points</th> </tr> </thead> <tbody> <tr> <td>1</td> <td></td> <td>EXAMPL-1.2.3</td> <td>K1</td> <td>1</td> </tr> <tr> <td>5</td> <td>c</td> <td>EXAMPL-4.5.6</td> <td>K2</td> <td>1</td> </tr> <tr> <td>6</td> <td>b, d</td> <td>EXAMPL-7.8.9</td> <td>K3</td> <td>2</td> </tr> </tbody> </table>						Question Number (#)	Correct Answer	Learning Objective (LO)	K-Level	Number of Points	1		EXAMPL-1.2.3	K1	1	5	c	EXAMPL-4.5.6	K2	1	6	b, d	EXAMPL-7.8.9	K3	2
Question Number (#)	Correct Answer	Learning Objective (LO)	K-Level	Number of Points																					
1		EXAMPL-1.2.3	K1	1																					
5	c	EXAMPL-4.5.6	K2	1																					
6	b, d	EXAMPL-7.8.9	K3	2																					

Answers																													
<table border="1"> <thead> <tr> <th>Question Number (#)</th> <th>Correct Answer</th> <th>Explanation / Rationale</th> <th>Learning Objective (LO)</th> <th>K-Level</th> <th>Number of Points</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>b</td> <td>The answer to life, the universe, and everything is a concept from Douglas Adams' science fiction series "The Hitchhiker's Guide to the Galaxy", where the supercomputer Deep Thought gives the answer 42.</td> <td>EXAMPL-1.2.3</td> <td>K1</td> <td>1</td> </tr> <tr> <td>5</td> <td>c</td> <td>The capital of France is Paris, known for art, fashion, and culture.</td> <td>EXAMPL-4.5.6</td> <td>K2</td> <td>1</td> </tr> <tr> <td>6</td> <td>b, d</td> <td>Dolphins and whales are classified as mammals because they are warm-blooded, breathe with lungs, and feed their young milk.</td> <td>EXAMPL-7.8.9</td> <td>K3</td> <td>2</td> </tr> </tbody> </table>						Question Number (#)	Correct Answer	Explanation / Rationale	Learning Objective (LO)	K-Level	Number of Points	1	b	The answer to life, the universe, and everything is a concept from Douglas Adams' science fiction series "The Hitchhiker's Guide to the Galaxy", where the supercomputer Deep Thought gives the answer 42.	EXAMPL-1.2.3	K1	1	5	c	The capital of France is Paris, known for art, fashion, and culture.	EXAMPL-4.5.6	K2	1	6	b, d	Dolphins and whales are classified as mammals because they are warm-blooded, breathe with lungs, and feed their young milk.	EXAMPL-7.8.9	K3	2
Question Number (#)	Correct Answer	Explanation / Rationale	Learning Objective (LO)	K-Level	Number of Points																								
1	b	The answer to life, the universe, and everything is a concept from Douglas Adams' science fiction series "The Hitchhiker's Guide to the Galaxy", where the supercomputer Deep Thought gives the answer 42.	EXAMPL-1.2.3	K1	1																								
5	c	The capital of France is Paris, known for art, fashion, and culture.	EXAMPL-4.5.6	K2	1																								
6	b, d	Dolphins and whales are classified as mammals because they are warm-blooded, breathe with lungs, and feed their young milk.	EXAMPL-7.8.9	K3	2																								

Figure 2: Three different ways to typeset question definitions in ISTQB Sample Exam Questions and Answers documents: a) a list of questions, b) an answer key, and c) a list of answers.

1 Question definitions

As an example of question definitions, I use the following YAML file named `questions.yml`:

```
num-questions: 3
max-score: 4
pass-score: 50 # percent
duration: [10, 15] # minutes
questions:
  1:
    learning-objective: 1.2.3
    k-level: K1
    number-of-points: 1
    question: >
      What is the answer to life,
      the universe, and everything?
    answers: {a: 24, b: 42, c: 64, d: 84}
    correct: [b]
    explanation: >
      The answer to life, the universe,
      and everything is a concept from
      Douglas Adams' science fiction
      series "The Hitchhiker's Guide to
      the Galaxy", where the supercomputer
      Deep Thought gives the answer 42.
  5:
    learning-objective: 4.5.6
    k-level: K2
    number-of-points: 1
    question: What's France's capital?
    answers: {a: Berlin, b: Madrid,
              c: Paris, d: Rome}
    correct: [c]
    explanation: >
      The capital of France is Paris,
      known for art, fashion, and culture.
  6:
    learning-objective: 7.8.9
    k-level: K3
    number-of-points: 2
    question: >
      Which two of the following animals
      are classified as mammals?
    answers: {a: Shark, b: Dolphin,
              c: Eagle, d: Whale,
              e: Crocodile}
    correct: [b, d]
    explanation: >
      Dolphins and whales are classified
      as mammals because they are
      warm-blooded, breathe with lungs,
      and feed their young milk.
```

The file specifies three questions. For each question, it provides up to five possible answers.

2 User interface

In this section, I show how we can use themes `istqb/sample-exam/questions`, and `/answers` to typeset the question definitions from the previous section.

2.1 Typesetting questions

As an example of an ISTQB Sample Exam Questions document, I use the following L^AT_EX file:

```
\documentclass{istqb}
\usepackage{markdown}
\markdownSetup {
  import = {
    istqb/sample-exam/questions =
    questions as qst
  }
}
\begin{document}
\istqbunnumberedsection{Questions}
\markdownInput[snippet=qst]{questions.yml}
\end{document}
```

The file imports the snippet `questions` from theme `istqb/sample-exam/questions` and uses it to:

1. Process question definitions in `questions.yml`.
2. Typeset the list of questions shown in Figure 2a.

2.2 Typesetting answer key and answers

As an example of an ISTQB Sample Exam Answers document, I use the following L^AT_EX file:

```
\documentclass{istqb}
\usepackage{markdown}
\markdownSetup {
  import = {
    istqb/sample-exam/answers = {
      answer-key as key,
      answers as ans,
    },
  }
}
\begin{document}
\istqblandscapebegin
\istqbunnumberedsection{Answer key}
\markdownInput[snippet=key]{questions.yml}
\istqbunnumberedsection{Answers}
\markdownInput[snippet=ans]{questions.yml}
\istqblandscapeend
\end{document}
```

The file imports snippets `answers` and `answer-key` from theme `istqb/sample-exam/answers` and uses them to:

1. Process question definitions in `questions.yml`.
2. Typeset the answer key shown in Figure 2b.
3. Typeset the list of answers shown in Figure 2c.

3 Implementation

In this section, I show the implementation of ISTQB Sample Exam Questions and Answers documents. To make programming easier, I use the high-level expl3 language in addition to plain TeX and L^AT_EX 2_&.

3.1 Processing question definitions

Both the snippet `questions` from the theme `istqb/ /sample-exam/questions` and the snippet `answers` from the theme `/answers` process question definitions before typesetting them. For the processing, they use the snippet `questions` from the theme `istqb/sample-exam`, which I describe in this section.

First, I define a key–value `istqb/questions`:

```

1  \keys_define:nn
2    { istqb / questions }
3    { num-questions .int_gset:N =
4      \g_istqb_num_questions_int,
5      max-score .int_gset:N =
6      \g_istqb_max_score_int,
7      pass-score .int_gset:N =
8      \g_istqb_pass_score_int }
```

The key–value stores the values in top-level unstructured fields `num-questions`, `max-score`, and `pass-score` from question definitions to variables.

Next, I define a key–value `istqb/questions/duration`:

```

9  \keys_define:nn
10   { istqb / questions / duration }
11   { 1 .int_gset:N =
12     \g_istqb_duration_min_int,
13     2 .int_gset:N =
14     \g_istqb_duration_max_int }
```

The key–value stores the values in the top-level structured field `duration` to variables.

Then, I define the snippet `questions` itself:

```

15 \seq_new:N \g_istqb_questions_seq
16 \markdownSetupSnippet
17   { questions }
18   { jekyllData,
19     expectJekyllData,
20     renderers = {
21       jekyllDataBegin = {
22         \seq_gclear:N
23         \g_istqb_questions_seq },
24       jekyllData(String|Number) = {
25         \keys_set:nn
26           { istqb / questions }
27           { { #1 } = { #2 } }},
28       jekyllDataMappingBegin = ,
29       jekyllDataSequenceBegin = {
30         \str_case:nn
31           { #1 }
32           { { duration } {
33             \markdownSetup
```

```

34           { code = \group_begin: ,
35             renderers = {
36               jekyllData(String
37                 |Number) = {
38                   \keys_set:nn
39                     { istqb / questions /
40                       duration }
41                     { { ##1 } = { ##2 } }},
42                   jekyllDataSequenceEnd =
43                     \group_end: }}}},
44             jekyllData(Mapping|Sequence)Begin += {
45               \str_case:nn
46                 { #1 }
47                 { { questions } {
48                   \markdownSetup
49                     { code = \group_begin: ,
50                       renderers = {
51                         jekyllData(Mapping
52                           |Sequence)End =
53                           },
54                         snippet = istqb
55                           / sample-exam / questions
56                           / list,
57                         renderers = {
58                           jekyllData(Mapping
59                             |Sequence)End
60                             += \group_end: }}}}}}}
```

The snippet processes question definitions as follows:

1. Define an empty sequence that will store question numbers.
2. Pass unstructured top-level fields to the key–value `istqb/questions`.
3. Pass the structured top-level field `duration` to the key–value `istqb/questions/duration`.
4. Pass the structured top-level field `questions` to a snippet `questions/list`.

Next, I define the snippet `questions/list`:

```

61 \markdownSetupSnippet
62   { questions / list }
63   { renderers = {
64     jekyllDataMappingBegin = {
65       \group_begin:
66       \tl_set:Nn
67         \l_istqb_current_question_tl
68         { #1 }
69       \seq_gput_right:NV
70         \g_istqb_questions_seq
71         \l_istqb_current_question_tl
72     \markdownSetup
73       { renderers = {
74         jekyllDataMappingEnd = },
75         snippet = istqb / sample-exam
76           / questions / *,
77         renderers = {
78           jekyllDataMappingEnd +=
79             \group_end: }}}}}}
```

The snippet processes each question as follows:

1. Store the current question number.
2. Pass all fields to a snippet `questions/*`.

Then, I define key-value `istqb/questions/*:`

```

80 \prop_new:N
81   \g_istqb_question_learning_objective_prop
82 \prop_new:N
83   \g_istqb_question_k_level_prop
84 \prop_new:N
85   \g_istqb_question_number_of_points_prop
86 \prop_new:N
87   \g_istqb_question_text_prop
88 \prop_new:N
89   \g_istqb_question_explanation_prop
90 \keys_define:nn
91   { istqb / questions / * }
92   { learning-objective .code:n = {
93     \prop_gput:cVn
94       { g_istqb_question_learning_objective
95         _prop }
96       \l_istqb_current_question_tl
97       { #1 } },
98   k-level .code:n = {
99     \prop_gput:NVn
100      \g_istqb_question_k_level_prop
101      \l_istqb_current_question_tl
102      { #1 } },
103   number-of-points .code:n = {
104     \prop_gput:cVn
105       { g_istqb_question_number_of_points
106         _prop }
107       \l_istqb_current_question_tl
108       { #1 } },
109   question .code:n = {
110     \prop_gput:NVn
111       \g_istqb_question_text_prop
112       \l_istqb_current_question_tl
113       { #1 } },
114   explanation .code:n = {
115     \prop_gput:NVn
116       \g_istqb_question_explanation_prop
117       \l_istqb_current_question_tl
118       { #1 } }}
```

The key-value stores the values in unstructured fields `number-of-points`, `learning-objective`, `k-level`, `explanation`, and `question` to dicts. The dicts use the current question number as the key.

Next, I define the snippet `questions/*:`

```

119 \markdownSetupSnippet
120   { questions / * }
121   { renderers = {
122     jekyllData(String|Number) = {
123       \keys_set:nn
124         { istqb / questions / * }
125         { { #1 } = { #2 } } },
126     jekyllDataSequenceBegin = {
127       \str_case:nn
```

```

128           { #1 }
129           { { correct } {
130             \markdownSetup
131               { code = \group_begin:, ,
132                 renderers = {
133                   jekyllDataSequenceEnd =
134                 },
135                   snippet = istqb
136                     / sample-exam / questions
137                     / * / correct,
138                   renderers = {
139                     jekyllDataSequenceEnd +=
140                       \group_end: }}}} },
141     jekyllDataMappingBegin = {
142       \str_case:nn
143         { #1 }
144         { { answers } {
145           \markdownSetup
146             { code = \group_begin:, ,
147               renderers = {
148                 jekyllDataMappingEnd = },
149                 snippet = istqb
150                   / sample-exam / questions
151                     / * / answers,
152                   renderers = {
153                     jekyllDataMappingEnd +=
154                       \group_end: }}}} }
```

The snippet processes question definitions as follows:

1. Pass unstructured fields to the key-value `istqb/questions/*`.
2. Pass the structured field `correct` to a snippet `questions/*/correct`.
3. Pass the structured field `answers` to a snippet `questions/*/answers`.

Notice the design pattern on lines 44–60, 64–79, and 126–154 that locally applies a `<snippet>` to an `<element>`.¹ This pattern redefines the renderer `<element>Begin`, which is placed to the output when the `<element>` starts, as follows:

1. Open a TeX group and apply the `<snippet>`.
2. Redefine the renderer `<element>End`, which is placed to the output when the `<element>` ends, so that it closes the TeX group.

Finally, I define snippets `questions/*/answers` and `/correct`:

```

155 \prop_new:N \g_istqb_answer_keys_prop
156 \prop_new:N \g_istqb_answers_prop
157 \seq_new:N \l_istqb_current_answer_keys_seq
158 \markdownSetupSnippet
159   { questions / * / answers }
```

¹ Such design patterns can be repetitive and difficult to understand without additional comments in the code. Markdown Enhancement Proposal (MEP) 445 [6] envisions support for higher-order snippets that would make it possible to hide such design patterns behind easy-to-read shorthands.

```

160 { renderers = {
161     jekyllData(String|Number) = {
162         \seq_put_right:Nn
163         \l_istqb_current_answer_keys_seq
164         { #1 }
165         \tl_set:NV
166         \l_tmpa_tl
167         \l_istqb_current_question_tl
168         \tl_put_right:Nn
169         \l_tmpa_tl
170         { / #1 }
171         \prop_gput:NVn
172         \g_istqb_answers_prop
173         \l_tmpa_tl
174         { #2 } },
175     jekyllDataMappingEnd += {
176         \clist_set_from_seq:NN
177         \l_istqb_current_answer_keys_clist
178         \l_istqb_current_answer_keys_seq
179         \prop_gput:NVv
180         \g_istqb_answer_keys_prop
181         \l_istqb_current_question_tl
182         { l_istqb_current_answer_keys
183             _clist } } }
184     \prop_new:N \g_istqb_answer_correct_keys_prop
185     \seq_new:N
186     \l_istqb_current_answer_correct_keys_seq
187     \markdownSetupSnippet
188     { questions / * / correct }
189     { renderers = {
190         jekyllData(String|Number) = {
191             \seq_put_right:cn
192             { l_istqb_current_answer_correct
193                 _keys_seq }
194             { #2 } },
195         jekyllDataSequenceEnd += {
196             \clist_set_from_seq:cc
197             { l_istqb_current_answer_correct
198                 _keys_clist }
199             { l_istqb_current_answer_correct
200                 _keys_seq } }
201         \prop_gput:NVv
202         \g_istqb_answer_correct_keys_prop
203         \l_istqb_current_question_tl
204         { l_istqb_current_answer_correct
205             _keys_clist } } }
```

The snippets accumulate potential and correct answer letters in a sequence, respectively. Then, they store the sequence as a comma-list to a dict that uses the current question number as the key.

Moreover, the snippet `questions/*/answers` stores potential answer texts to a dict that uses `(current question number)/(answer letter)` as key.

Notice that I used no format-specific code in this section. Therefore, I can use the theme `istqb/sample-exam` with any format that supports `expl3` such as plain TeX and ConTeXt, not just with L^AT_EX.

3.2 Typesetting questions

In this section, I describe the snippet `questions` from theme `istqb/sample-exam/questions`. This snippet typesets the list of questions in Figure 2a.

First, I import the theme `istqb/sample-exam` and I use the snippet `questions` from this theme to process question definitions:

```

1 \markdownSetup
2   { import = istqb / sample-exam }
3 \markdownSetupSnippet
4   { questions }
5   { snippet = istqb / sample-exam
6     / questions,
```

After the question definitions have been processed, I iterate over all question numbers. For each question number, I define a variable with code that typesets the corresponding question:

```

7   renderers = {
8     jekyllDataEnd = {
9       \seq_map_inline:Nn
10      \g_istqb_questions_seq
11      { \tl_set:Nn
12        \l_istqb_question_tl
13        {
```

First, I add a section heading for the question:

```

14   \tl_set:Nn
15     \l_tmpa_tl
16     { Question~\# ##1~( }
17   \prop_get:cnN
18     { g_istqb_question_number
19       _of_points_prop }
20     { ##1 }
21     \l_tmpb_tl
22     \tl_put_right:NV
23     \l_tmpa_tl
24     \l_tmpb_tl
25     \tl_put_right:Nn
26     \l_tmpa_tl
27     { ~Point }
28     \int_compare:VNnF
29     \l_tmpb_tl = { 1 }
30     { \tl_put_right:Nn
31       \l_tmpa_tl
32       { s } }
33     \tl_put_right:Nn
34     \l_tmpa_tl
35     { ) }
36     \exp_args:NNV
37     \subsection *
38     \l_tmpa_tl
39     \exp_args:NVV
40     \markboth
41     \l_tmpa_tl
42     \l_tmpa_tl
43     \exp_args:NnnV
44     \addcontentsline
```

```

45   { toc }
46   { subsection }
47   \l_tmpa_t1

```

Next, I add the question text and potential answers:

```

48   \prop_item:Nn
49     \g_istqb_question_text_prop
50     { ##1 }
51   \prop_get:NnN
52     \g_istqb_answer_keys_prop
53     { ##1 }
54     \l_tmpa_clist
55   \begin{enumerate}
56     \clist_map_inline:Nn
57       \l_tmpa_clist
58       { \item [ ####1 ] }
59       \prop_item:Nn
60         \g_istqb_answers_prop
61         { ##1 / ####1 } }
62   \end{enumerate}
63   \medskip

```

Lastly, I add the text “Select *⟨number of correct answers⟩* option(s).”:

```

64   \prop_get:cNn
65     { g_istqb_answer_correct
66       _keys_prop }
67     { ##1 }
68     \l_tmpa_clist
69   \int_set:Nn
70     \l_tmpa_int
71     { \clist_count:N
72       \l_tmpa_clist }
73   Select~\int_case:nn
74     { \l_tmpa_int }
75     { { 1 } { ONE~option }
76       { 2 } { TWO~options } }
77 }

```

Finally, I typeset the code from the variable at natural height and store the result to a vertical box:

```

78   \vbox_set:NV
79     \l_tmpa_box
80     \l_istqb_question_t1

```

For short questions, I insert the box to the current list for typesetting to prevent page breaks within the question. For longer questions, I place the content of the variable to the input stream, so that page breaks can occur naturally:

```

81   \dim_compare:nNnTF
82     { \box_ht:N \l_tmpa_box }
83     >
84     { 0.3 \paperheight }
85     { \tl_use:N
86       \l_istqb_question_t1 }
87     { \box_use:N \l_tmpa_box }
88   \par }}}}

```

3.3 Typesetting the answer key

In this section, I describe the snippet `answer-key` from the theme `istqb/sample-exam/answers`. This snippet typesets the answer key in Figure 2b.

First, I load packages `multicol` and `supertabular`:

```

1  \RequirePackage { multicol }
2  \RequirePackage { supertabular }
3  \RequirePackage { array }
4  \newcolumntype
5    { C }
6    [ 1 ]
7    { >{ \centering\arraybackslash } p { #1 } }

```

The packages allow me to typeset the answer key as a table in a two-column layout that automatically inserts column breaks.

Next, I import the theme `istqb/sample-exam` and I use the snippet `questions` from this theme to process question definitions:

```

8  \markdownSetup
9    { import = istqb / sample-exam }
10 \markdownSetupSnippet
11   { answer-key }
12   { snippet = istqb / sample-exam
13     / questions,

```

After the question definitions have been processed, I start a two-column layout:

```

14   renderers = {
15     jekyllDataEnd = {
16       \begin{multicols} { 2 }

```

Then, I set the heading and the tail of the table:

```

17   \tablehead
18     { \hline
19       \textbf
20         { Question-Number~(\#) } &
21       \textbf
22         { Correct-Answer } &
23       \textbf
24         { Learning-Objective~(LO) } &
25       \textbf
26         { K-Level } &
27       \textbf
28         { Number-of-Points } \\ }
29   \tabletail { \hline }
30   \tablelasttail { \hline }

```

Next, I define a variable that typesets the table:

```

31   \tl_set:Nn
32     \l_istqb_answer_key_table_t1
33   {

```

First, I start the table:

```

34   \begin{supertabular}
35     { | C { 1.9cm } | C { 1.5cm } }
36     { | C { 2.4cm } | C { 1.4cm } }
37     { | C { 1.9cm } | } }
38

```

Next, I iterate over all question numbers:

```
39      \seq_map_inline:Nn
40          \g_istqb_questions_seq
41      {
42          \tl_put_right:Nn
43              \l_istqb_answer_key_table_tl
44          { \hline }
```

For each question, I add the question number:

```
45      \tl_put_right:Nn
46          \l_istqb_answer_key_table_tl
47          { \textbf{\{ \#1 \}} & }
```

Next, I add the correct answer letters:

```
48      \prop_get:cnN
49          { g_istqb_answer_correct
50              _keys_prop }
51          { \#1 }
52          \l_tmpa_clist
53      \tl_put_right:Ne
54          \l_istqb_answer_key_table_tl
55          { \clist_use:Nn
56              \l_tmpa_clist
57          { ,~ } & }
```

Then, I add the learning objective:

```
58      \tl_put_right:NV
59          \l_istqb_answer_key_table_tl
60          \g_istqb_prefix_tl
61      \tl_put_right:Nn
62          \l_istqb_answer_key_table_tl
63          { - }
64      \prop_get:cnN
65          { g_istqb_question_learning
66              _objective_prop }
67          { \#1 }
68          \l_tmpa_tl
69      \tl_put_right:NV
70          \l_istqb_answer_key_table_tl
71          \l_tmpa_tl
72      \tl_put_right:Nn
73          \l_istqb_answer_key_table_tl
74          { & }
```

Next, I add the K-level:

```
75      \prop_get:NnN
76          \g_istqb_question_k_level_prop
77          { \#1 }
78          \l_tmpa_tl
79      \tl_put_right:NV
80          \l_istqb_answer_key_table_tl
81          \l_tmpa_tl
82      \tl_put_right:Nn
83          \l_istqb_answer_key_table_tl
84          { & }
```

Lastly, I add the number of points:

```
85      \prop_get:cnN
86          { g_istqb_question_number
87              _of_points_prop }
88          { \#1 }
```

```
89          \l_tmpa_tl
90      \tl_put_right:NV
91          \l_istqb_answer_key_table_tl
92          \l_tmpa_tl
93      \tl_put_right:Nn
94          \l_istqb_answer_key_table_tl
95          { \\"{} }
```

After I have iterated over all question numbers, I end the table, I place the content of the variable to the input stream, and I end the multicolumn layout:

```
97      \tl_put_right:Nn
98          \l_istqb_answer_key_table_tl
99          { \end{supertabular} } }
100     \tl_use:N
101         \l_istqb_answer_key_table_tl
102     \end{multicols} } }}
```

3.4 Typesetting answers

In this section, I describe the snippet `answers` from the theme `istqb/sample-exam/answers`. This snippet typesets the list of answers in Figure 2c.

First, I load package `longtable`:

```
1 \RequirePackage{longtable}
2 \dim_const:Nn
3     \c_explanation_width_dim
4     { 11.15cm }
```

The package allows me to typeset the list of answers as a table that automatically inserts page breaks.

Next, I use the snippet `questions` from theme `istqb/sample-exam` to process question definitions:

```
5 \markdownSetupSnippet
6 { answers }
7 { snippet = istqb / sample-exam
8     / questions,
```

After the question definitions have been processed, I define a variable that typesets the table:

```
9 renderers = {
10     jekyllDataEnd = {
11         \group_begin:
12             \tl_set:Nn
13                 \l_istqb_answers_table_tl
14             { }
```

First, I start the table and I set its heading:

```
15         \begin
16             { longtable }
17             { | C { 1.9cm } | C { 1.5cm } }
18             | p
19                 { \c_explanation_width_dim }
20                 | C { 2.4cm } | C { 1.4cm }
21                 | C { 1.9cm } | }
22         \hline
23         \textbf{
24             { Question-Number~(\#) } &
25             \textbf{Correct-Answer} } &
```

```

26   \multicolumn
27     { 1 }
28     { C
29       { \c_explanation_width_dim }
30       | }
31     { \textbf
32       { Explanation~/~Rationale } }
33     } &
34   \textbf
35     { Learning~Objective~(LO) } &
36   \textbf { K-Level } &
37   \textbf { Number~of~Points } \\
38   \hline
39   \endhead

```

Next, I iterate over all question numbers:

```

40   \seq_map_inline:Nn
41     \g_istqb_questions_seq
42   {

```

For each question, I add the question number:

```

43     \tl_put_right:Nn
44       \l_istqb_answers_table_tl
45     { \textbf
46       { ##1 }
47     \addcontentsline
48       { toc }
49       { subsection }
50       { Question~\# ##1 } & }

```

Next, I add the correct answer letters:

```

51   \prop_get:cnN
52     { g_istqb_answer_correct
53       _keys_prop }
54     { ##1 }
55     \l_tmpa_clist
56   \tl_put_right:Ne
57     \l_istqb_answers_table_tl
58   { \clist_use:Nn
59     \l_tmpa_clist
60     { ,~ } & }

```

Then I add the explanation text:

```

61   \tl_put_right:Nn
62     \l_istqb_answers_table_tl
63   { \begin
64     { minipage }
65     [ t ]
66     \c_explanation_width_dim }
67   \prop_get:cnN
68     { g_istqb_question_explanation
69       _prop }
70     { ##1 }
71     \l_tmpa_tl
72   \tl_put_right:NV
73     \l_istqb_answers_table_tl
74     \l_tmpa_tl
75   \tl_put_right:Nn
76     \l_istqb_answers_table_tl
77   { \end { minipage } }

```

```

78           \medskip }
79   \tl_put_right:Nn
80     \l_istqb_answers_table_tl
81   { & }

```

Next, I add the learning objective:

```

82   \tl_put_right:NV
83     \l_istqb_answers_table_tl
84     \g_istqb_prefix_tl
85   \tl_put_right:Nn
86     \l_istqb_answers_table_tl
87   { - }
88   \prop_get:cnN
89     { g_istqb_question_learning
90       _objective_prop }
91     { ##1 }
92     \l_tmpa_tl
93   \tl_put_right:NV
94     \l_istqb_answers_table_tl
95     \l_tmpa_tl
96   \tl_put_right:Nn
97     \l_istqb_answers_table_tl
98   { & }

```

Then, I add the K-level:

```

99   \prop_get:NnN
100    \g_istqb_question_k_level_prop
101    { ##1 }
102    \l_tmpa_tl
103   \tl_put_right:NV
104     \l_istqb_answers_table_tl
105     \l_tmpa_tl
106   \tl_put_right:Nn
107     \l_istqb_answers_table_tl
108   { & }

```

Lastly, I add the number of points:

```

109  \prop_get:cnN
110    { g_istqb_question_number_of
111      _points_prop }
112    { ##1 }
113    \l_tmpa_tl
114   \tl_put_right:NV
115     \l_istqb_answers_table_tl
116     \l_tmpa_tl
117   \tl_put_right:Nn
118     \l_istqb_answers_table_tl
119   { \\ \hline } }

```

After I have iterated over all question numbers, I end the table and I place the content of the variable to the input stream:

```

120   \tl_put_right:Nn
121     \l_istqb_answers_table_tl
122   { \end { longtable } }
123   \tl_use:N
124     \l_istqb_answers_table_tl
125   \group_end: } }

```

Conclusion

In this article, I have demonstrated the practical application of Markdown themes through a project that enabled the International Software Testing Qualifications Board (ISTQB) to produce their certification study materials from Markdown and YAML sources. While my previous article [5] focused on the underlying concepts of Markdown themes, this article provides concrete code used in a real-world software project. I hope this practical demonstration raises awareness of Markdown themes and illustrates how users can incorporate them into their own projects.

For ISTQB, the project has yielded numerous benefits: Writing text in a structured format using Markdown and YAML, while generating visually appealing outputs with L^AT_EX, facilitates the separation of content from formatting. This ensures consistent application of the document's visual style across all ISTQB content. Additionally, the structured text enables content verification against YAML schemas and ISTQB writing rules and allows for the creation of a complex knowledge base through automated processing. This enhances the quality of learning materials and reduces administrative overhead.

Moreover, the plain text formats of Markdown and YAML offer significant advantages over binary formats like Microsoft Office. They allow for efficient version control, better tracking of changes, collaborative editing, and fewer defects in the final products. The capability to produce various output formats, such as EPUB, HTML, and PDF with functional hyperlinks and cross-references, further amplifies the utility of this approach.

Related work

In my approach, I developed an event-based L^AT_EX parser that constructs and typesets expl3 data structures that represent YAML files.² My approach works in any T_EX engine with shell access, such as pdfT_EX and X_ET_EX, not just LuaT_EX.

In the previous issue of *TUGboat* [4], Erik Nijenhuis showed a different approach towards typesetting YAML files in L^AT_EX. In their approach, Erik used their lua-placeholders library [3] to load YAML files into Lua tables and then query them from T_EX code. Erik's approach requires LuaT_EX but can be more convenient for non-programmers.

² My focus on processing and typesetting YAML files may seem contrary to the title of this article “Markdown themes in practice”. However, authors may use Markdown markup in YAML files. In the examples from this article, we might use Markdown to format questions, answers, and explanations.

Both Erik's and my approaches use the tinyyaml Lua library [2]. LuaT_EX users who are interested in processing YAML files directly from Lua code may find it convenient to use tinyyaml directly.

Acknowledgements

I wish to extend my special thanks to Tereza Vrabcová, Marei Peischl, Daniel Polan, and Petr Sojka for their invaluable insights and thorough review of my work. Their expertise and thoughtful feedback have been instrumental in shaping the final manuscript.

I would also like to thank Greg at fiverr.com/quickcartoon for their illustrations of the wolf mascot, which have provided an engaging visual identity of the Markdown package over the past four years.

References

- [1] ISTQB.ORG. L^AT_EX+Markdown template, 2024. github.com/istqborg/istqb_product_base
- [2] Z. Lee. lua-tinyyaml: A tiny YAML (subset) parser for pure Lua, 2023. ctan.org/pkg/lua-tinyyaml
- [3] E. Nijenhuis. lua-placeholders: Specifying placeholders for demonstration purposes, 2024. ctan.org/pkg/lua-placeholders
- [4] E. Nijenhuis. Specifying and populating documents in YAML with lua-placeholders in L^AT_EX. *TUGboat* 45(1):65–76, 2024. doi.org/10.47397/tb/45-1/tb139nijenhuis-placeholders
- [5] V. Novotný. Markdown 2.10.0: L^AT_EX themes & snippets, two flavors of comments, and luametaT_EX. *TUGboat* 42(2):186–193, 2021. doi.org/10.47397/tb/42-2/tb131novotny-markdown
- [6] V. Starý Novotný. Parametric snippets, 2024. github.com/Witiko/markdown/issues/445

◇ Vít Starý Novotný
Studená 453/15
Brno 63800, Czech Republic
witiko (at) mail dot muni dot cz
github.com/witiko

A large-scale format compliance checker for T_EX Font Metrics

Didier Verna

Abstract

We present `tfm-validate`, a T_EX Font Metrics format checker. The library's core functionality is to inspect TFM files and report any discovered compliance issue. It can be run on individual files or complete directory trees. `tfm-validate` also provides a convenience function to (in)validate a local T_EX Live installation. When run this way, the library processes every TFM file in the distribution and generates a website aggregating all the discovered non-compliance issues. One public instance of `tfm-validate` is now automatically triggered on a daily basis. The corresponding website is available at texlive.info/tfm-validate/.

1 Introduction

As part of ETAP,¹ our experimental typesetting algorithms platform [8, 9], we have developed a parser for TFM (T_EX Font Metrics) files, simply called `tfm`. To ensure robustness, a parser for an official data format must be prepared to handle all sorts of compliance problems, with varying degrees of seriousness ranging from simple warnings to non-recoverable errors. `tfm` not only provides a rich (hopefully exhaustive) ontology of errors, but also a powerful recovery mechanism, allowing for proceeding as long as possible with the parsing, for example by fixing errors on the fly or discarding problematic input.

A side-effect of `tfm`'s robustness is that it is possible to use it as a validation tool rather than for loading font information. Indeed, the `tfm` exception handler reifies the problematic situations into objects (in the “Object-Oriented” sense) which can be silently collected until the parsing is over or needs to be terminated prematurely. These objects can in turn be used to produce a full compliance report for the analyzed file. We have automated this process for the whole T_EX Live distribution, resulting in the (in)validation of almost 80 000 fonts, and the generation of a website providing direct access to the generated compliance reports.

This paper is organized as follows. Section 2 provides an overview of the `tfm` library and explains how it is made robust. Section 3 describes the very peculiar exception handling mechanism in use, and how it simplifies the design of `tfm-validate` considerably. Finally, Section 4 analyzes the results of

the TFM validation process applied to the whole T_EX Live distribution.

2 The `tfm` library

The `tfm`² library was designed to bring T_EX Font Metrics information to Common Lisp [1] applications. Essentially, it provides an entry point function called `load-font`, which takes a file name as argument and returns a data structure containing an abstract representation of the contents of the TFM file. A full description of the library is beyond the scope of this paper. The interested reader will find a complete user manual in the distribution, as well as online. The important thing for this discussion is that `tfm` aims at being both robust and flexible.

2.1 Robustness

Robustness for a parser means that it should be prepared to handle *all* the possible problematic situations, for example in order to abort loading the culprit font file and exit gracefully, rather than just crashing or behaving erratically. During the development of `tfm`, we have identified twenty such situations, with varying degrees of severity.

Examples of critical situations include truncated files or invalid section pointers, making it impossible to know exactly where to find character, ligature, kerning information, *etc.* In those situations, there is nothing clever one can do to make the bogus font functional.

A less critical, yet problematic situation, would be the detection of a cycle in a ligature program, resulting in an infinite loop when attempting the ligature. In such a case, we can still hope to get a functional (although incomplete) font if we just forget about the ligature(s).

Non-critical situations might be inconsistencies in parts of the TFM file which are purely informative (such as several places in the header) and not used to render the font. T_EX itself simply ignores a number of such situations and proceeds normally.

Finally, note that the severity of a problem may depend on the context. One interesting such case is that of the font's design size. The TFM format requires it to be greater than 1. At the same time, T_EX allows the design size to be overridden by the user (this is what happens when you say `\font\foo=cmr10 at 12pt` for example). An invalid design size is normally an error, but it doesn't really hurt when overridden by a correct one. Hence, the `tfm` library signals an error in the former case, but only a warning in the latter.

¹ github.com/didierverna/etap

² github.com/didierverna/tfm

```

CL-USER> (tfm:load-font "/tmp/cmr10.tfm")
While reading /tmp/cmr10.tfm,
while reading the character encoding scheme string,
padded string "TeX (ex)" is not in BCPL format.
See §10 of the TFtoPL documentation for more information.
[Condition of type NET.DIDIERVERNA.TFM:INVALID-PADDED-STRING]

Restarts:
0: [KEEP-STRING] Keep it anyway.
1: [FIX-STRING] Fix it using /'s and ?'s.
2: [DISCARD-STRING] Discard it.
3: [CANCEL-LOADING] Cancel loading this font.
--more--

```

Figure 1: Sample interactive recovery session

2.2 Flexibility

Flexibility for a parser means that *when possible*, it should provide less drastic ways to recover from problems than just giving up. `tfm` currently provides a dozen recovery options, the availability of which depends on the situation.

As mentioned previously, it is possible to discard a ligature or a kerning instruction rather than aborting the whole loading process if something is wrong (like an invalid character code). Another example is the requirement that the width, height, depth and italic corrections tables all start with a first value of 0. When appropriate, `tfm` offers to fix a bogus value (by setting it to 0) and proceed, rather than just aborting.

The question of whether a font would be functional after recovery is crucial. Discarding a single ligature because of an invalid character code may be safe. Resetting a non-zero first table entry may be safe as well, but it might also be the case that the entire table (or the whole font for all we know) is in fact completely corrupted. The point here is that it is not the job of the library to make a decision, only to offer options.

In fact, having options may come in handy for interactive use (Common Lisp applications can be run both interactively and as standalone executables). Figure 1 illustrates this. In this example, a fake `cmr10` font has been corrupted on purpose: the character encoding string present in the file's header has been modified to contain parentheses, which is illegal. When loading the font interactively, the user ends up in the debugger and is presented with a number of “soft” recovery options (keeping the string as-is, fixing it, discarding it), in addition to plain cancellation.

A non-interactive application, on the other hand, would have the ability to automatically select an option without requiring user intervention. In production, the most likely choice is `CANCEL-LOADING` (and then fall back to another font). Given the goal we are trying to achieve here however, we would prefer to select the recovery option that allows us to proceed with the parsing for as long as possible.

Figure 2 summarizes all the possible problems (rectangles) and the corresponding recovery options (ellipses) that `tfm` provides. The details are not important. The intent of this picture is to convey the feeling that even for a relatively simple file format, a complete error/recovery ontology can quickly become rather intricate.

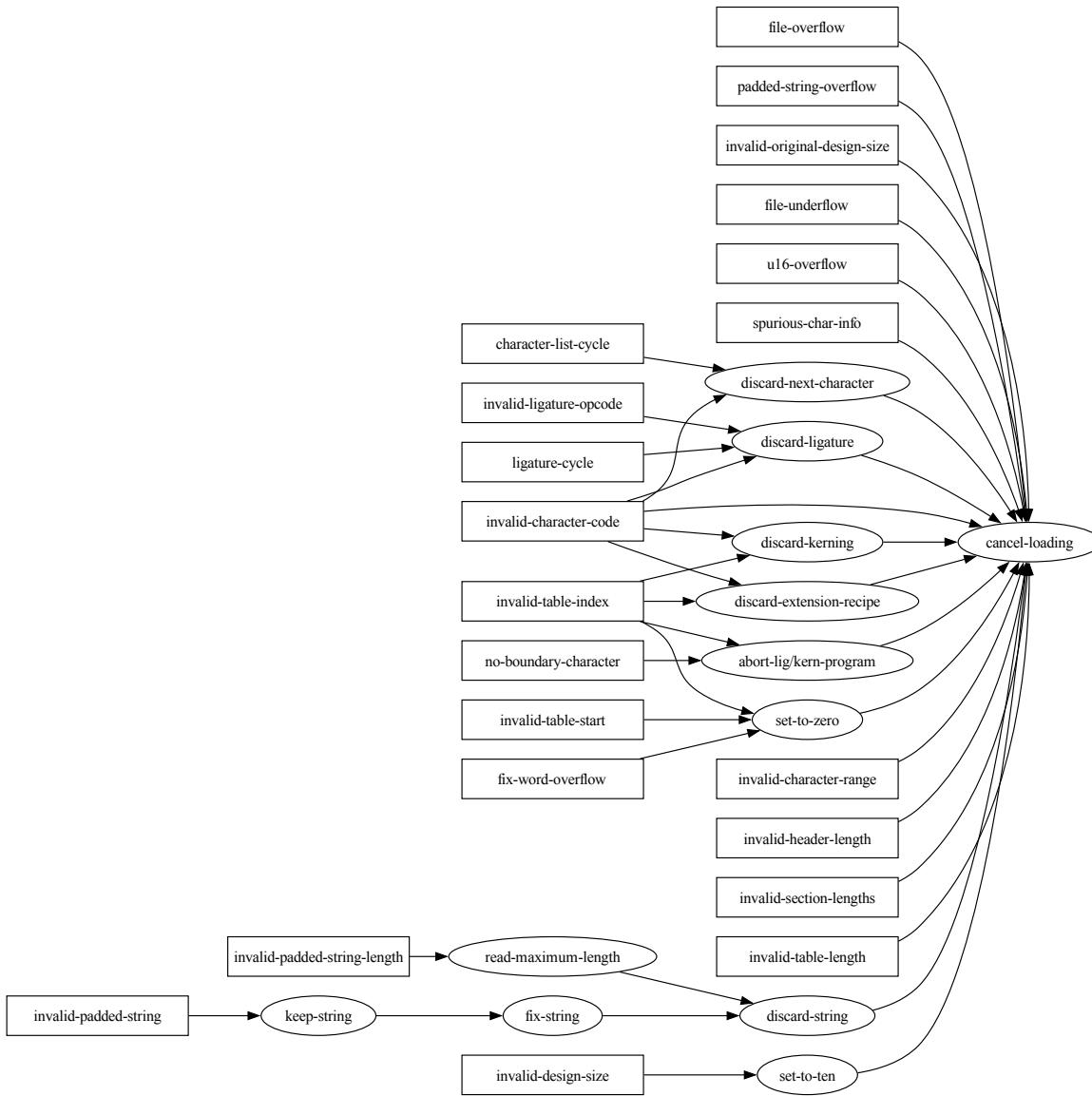
3 The `tfm-validate` library

While `tfm` was originally a requirement for ETAP, `tfm-validate`³ is a typical case of a project that was born out of curiosity rather than necessity, and also because it was quite easy to do. The key ingredient in `tfm-validate`'s design simplicity is the very peculiar exception handling that Common Lisp provides, the so-called “condition system” [4, 6], which we'll now describe.

3.1 The Common Lisp condition system

Most programming languages with explicit support for exception handling use some form of “try/catch” mechanism, as illustrated in the left part of Figure 3. A program may establish points at which exceptions (thrown elsewhere) are caught and handled. In the example, the program throws an exception while executing `func4`. The exception travels up the call stack until it reaches the handler in `func2`. If the exception is caught there, execution resumes at that

³ github.com/didierverna/tfm-validate

**Figure 2:** The `tfm` error ontology

point. Otherwise, the exception goes one more step up, to `func1`.

Unfortunately, this mechanism suffers from an unnecessary limitation in expressiveness: the exception handler actually does two different things at the same time (and for no good reason). Namely, a control point established by a handler serves not only to catch an exception, but also to resume execution. There is in fact no reason to limit ourselves to such a simple scheme, and the Common Lisp condition system adds one more degree of freedom to its exception handling infrastructure.

3.1.1 Signal / Handle / Restart

The equivalent of “throwing an exception” is called “signalling a condition” in Lisp, and the concept is equivalent. There is, however, no such thing as single catch/resume points in the Lisp condition system. Instead, a program establishes points where it is possible to resume execution (called “restarts”), and points where conditions are caught (called “handlers”). This is illustrated in the right part of Figure 3. Given the same scenario as before, `func4` signals a condition. The condition goes up the call

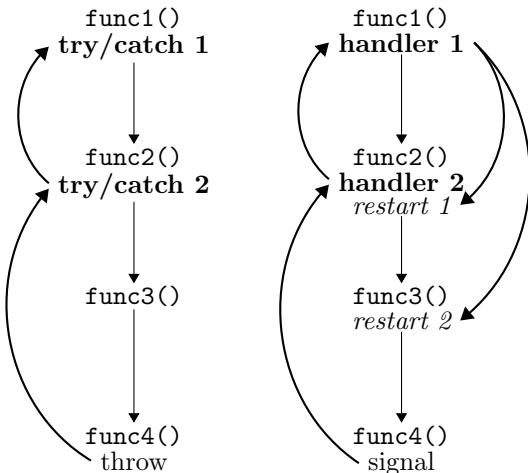


Figure 3: try/catch vs. handle/restart

stack and finds a handler in `func2`. If this handler is interested, it now has two options: resume execution right here (with *restart 1*), or in `func3` with *restart 2*. Otherwise, the condition goes one more step up and the handler in `func1` is given the same two choices, since no additional restart is installed.

3.1.2 First-class conditions

A second important aspect of the Common Lisp condition system (not unique to Lisp this time) is that it is grounded in CLOS [5], the object-oriented layer of the language. This means that creating an ontology of errors boils down to designing a hierarchy of condition *classes*, and the signalled conditions are reified as *objects*, that is, instances of the corresponding classes. In other words, conditions are “first-class” citizens in the language [2, 7].

3.2 The design simplicity of `tfm-validate`

Why is all this relevant to the design simplicity of `tfm-validate`? As mentioned before (Section 2.2), it is not the job of `tfm` to handle errors; only to detect them and offer as many soft recovery options as possible, for flexibility. In the technical terms of the Common Lisp condition system, we now understand that `tfm` *sends* conditions and provides a variety of *restarts*, but does *not* establish any handlers.

Short of handling conditions, a `tfm` user ultimately ends up in the debugger if something goes wrong (again, as demonstrated in Figure 1). But the key point is that since restarts and handlers are different concepts, it is possible to decide what to do programmatically rather than interactively, by establishing handlers *outside* `tfm`, or more specifically *around* calls to it.

We can now understand why `tfm-validate` was in fact quite easy to write. The main entry

point is a function called `invalidate-font`, which calls `tfm`'s `load-font` function. But before doing so, `invalidate-font` establishes a (rather large) handler for all the conditions that `tfm` may signal, and for every one of them, selects the “softest” restart available, allowing to proceed with the parsing for as long as possible. Note again that because handlers and restarts are not required to be located at the same places in the code, no modifications to the original `tfm` library are required to make it work like a compliance checker rather than for loading fonts.

But `invalidate-font` doesn't stop there. Every time a condition is caught, the function collects it before restarting (remember that conditions are actual objects). The return value of `invalidate-font` is thus the list (possibly empty) of all the signalled conditions. In fact, `invalidate-font` doesn't do any printing by itself. After execution, the user gets the list of signalled conditions, and is then free to do whatever they wish with it, such as inspecting, printing in one form or another, or even generating a website ...

4 TeX Live validation

... which is the point we are getting to. The function `invalidate-font` which, again, is essentially a wrapper around `load-font`, collecting the signalled conditions, is 68 lines long. With 10 more lines, we offer a function checking the compliance of a whole directory tree rather than of a single font file. This function is unsurprisingly called `invalidate-directory`.

At that point, we were curious about the state of the TeX Live distribution, since it is a rather large repository of TFM files, all located under a single directory tree. As it turns out, running our function `invalidate-directory` on it revealed a quite large number of non-compliance issues, which was an incentive to put all that information into a human-readable shape.

4.1 Non-compliance reports

The `tfm-validate` library provides yet another entry point called `invalidate-texlive`. It generates a website aggregating non-compliance reports (one HTML page per culprit TFM file) plus a couple of indexes. With the help of Norbert Preining, the system is now run on a daily basis and the corresponding website is made available at texlive.info/tfm-validate/.

At the time of this writing, the results of the validation process are as follows. 79016 fonts are inspected. 2983 fonts are skipped because `tfm` doesn't support OFM or JFM yet. 770 fonts are found to be non-compliant, which may seem quite a lot. On the

other hand, there are only 4 kinds of problems: 3 of which are considered warnings, and only a single one a truly unrecoverable error.

4.2 File overflow

By far, the most common issue that `tfm-validate` finds is file overflows, affecting 628 fonts. The TFM standard mandates that the first two bytes of a TFM file encode the file's length. A “file overflow” warning is signalled if the actual file's length is greater than expected. Note that `tfm` knows about the special values 0, 9, and 11, denoting extended TFM files (OFM or JFM), which are not supported yet.

Of course, when the declared file size disagrees with the actual, there is no way to tell for sure which (if any) is correct. However, absent any other problem during parsing, the file containing a tail of junk is much more likely than the first two bytes (only) being corrupted, hence a warning.

A quick test on a couple of such files seems to confirm that hypothesis. We compiled a sample document with them, and it appears that not only `TEX` has no problem loading the fonts, the outputs look normal as well. On top of that, let us mention that `tftopl` adopts the same posture: it signals the problem but otherwise just discards the junk (Section 20 of `tftopl`).

Further investigation on the tails was inconclusive. In particular we couldn't figure out whether some tails contain meaningful information rather than just junk (a possible cause for file overflows could be padding to storage blocks). As a consequence, the signalled warnings do not include the tails' content.

4.3 String overflow

The situation is slightly different with the next kind of problem we encountered, namely, padded string overflows, currently affecting 74 fonts.

A TFM file may contain two optional strings in its header. The first one, 40 bytes long, identifies the character coding scheme. The second one, 20 bytes long, is the font identifier (font family name). These strings are supposed to be in BCPL format. In particular, the first byte must contain the actual length of the string.

`tfm` signals a “padded string overflow” warning when a BCPL string is not padded with zeros. Doug McKenna suggested⁴ that padding a BCPL string with zeros may not have always been a requirement, as it was only added to `pltotf` in April 1983, for version 1.3, that is, two years after its initial release (Section 87 of `pltotf`). On the other hand, David

Fuchs mentioned padding with zeros as early as in February 1981 [3].

Anyway, the decision as to whether a padded string overflow should be a warning or an error is even simpler to make than in the case of a file overflow. Those strings are purely informative, they have no impact on the font's usability, so it does not hurt to continue loading the font.

Besides, the padding area seems to have been intentionally abused in the majority of the cases: a lot of fonts contain “Y&Y Inc” in there, making their origin quite clear. Because of that (and contrary to file overflows), the content of the padding area is included in the warnings.

4.4 Spurious char info

The next problem we encountered (also a warning, affecting 66 files) is a more obscure matter. TFM files have a so-called “char info table” providing the actual character metrics of the font. The table contains 4-byte entries for the full range of characters from the minimum character code (*bc*) to the maximum one (*ec*). However, a font may also have “holes” in this range, that is, undefined characters for some codes between *bc* and *ec*.

Undefined characters must have a width of 0, materialized by a width table index of 0 as well. The spurious char info warning indicates that an entry for a non-existent character is not completely zeroed out. In the problematic char info entries that we found, the third byte usually has a value of 1 (indicating an index into a ligature or kerning program), and sometimes a non-zero fourth byte (the actual index).

A possible explanation would have been the existence of a so-called “boundary character” (also an obscure matter in TFM) which is not required to exist for real in the font, but upon inspection of several problematic ones, this appears not to be the case.

`tftopl` completely ignores characters with a width index of 0 (Section 78 of `tftopl`), and `pltotf` zeroes out non-existent characters (Section 74 of `pltotf`). All the more reasons to not consider this problem a showstopper.

4.5 Fix word overflow

Finally, this one is the only true error we encountered, and it only affects two fonts: `ArevSans-Bold`, and `ArevSans-BoldOblique`. TFM has a notion of “fix word” numerical values which (with two exceptions) must remain within $] -16, +16 [$. In particular, the actual font metrics (width, height, depth, and italic correction) are expressed in fix words.

⁴ reference lost; could have been in a thread on `texhax...`

In the two aforementioned fonts, exactly 124 such values are off the charts. Again, for the sake of flexibility, `tfm` offers a soft recovery option for this problem (see Figure 2): setting the culprit value to 0, which would most likely result in an unreadable document. `TeX` refuses to load these fonts, which confirms the severity of the problem; hence an error.

5 Related work

Manuel Pégourié-Gonnard wrote a Perl script⁵ for checking the validity of a variety of files using external programs (typically, `tftopl` for TFM files). It is our understanding that this script produces a somewhat terse output: it prints a list of “bad” files without collecting more specific information, let alone presenting it in a human readable form.

According to a comment by Karl Berry, the script took a long time to run and maintenance of the list of broken fonts was tedious, with no particular action happening on the part of the font maintainers to fix the problems, so using it was abandoned in August 2019.

6 Conclusion and perspectives

As mentioned before, this project was born out of curiosity rather than necessity, and because it was easy to develop. Whether it is actually useful remains to be seen. Perhaps having compliance problems publicly advertised on a website will be a new kind of incentive for authors to update their files, and perhaps this project will be more helpful to watch over new additions rather than blame older content.

One merit of this project is to provide an insight into the global status of TFM compliance over a large set of fonts. In particular, we can see that the surprising number of non-compliant files is mitigated by the fact that most issues are in fact benign (only two fonts were found to be truly unusable).

In the future, we plan on adding new font formats to the system. Provided that we can find the appropriate documentation, OFM and JFM are likely to be straightforward additions and as a matter of fact, the `tfm` library is already prepared for it. We have also started to work on an OTF parser, designed along the same lines (that is, built around the Common Lisp condition system) but this will take slightly longer to complete.

Finally, the current layout of the website still has a lot of room for improvements. It currently provides two indexes, but the general question boils down to offering different forms of access to cross-referenced information. Karl Berry has already suggested a cou-

ple of possible ways to do so, which we will definitely take into account in the future.

Acknowledgements

The author wishes to thank Norbert Preining, Karl Berry, and Doug McKenna for fruitful exchanges during the development of both `tfm` and `tfm-validate`.

References

- [1] ANSI. American National Standard: Programming Language—Common Lisp. ANSI X3.226:1994 (R1999), 1994.
- [2] R. Burstall. Christopher Strachey—Understanding programming languages. *Higher Order Symbolic Computation*, 13(1–2):51–55, 2000.
- [3] D. Fuchs. `TeX` font metric files. *TUGboat*, 2(1):12–16, Feb. 1981. tug.org/TUGboat/tb02-1/tb02fuchstfm.pdf
- [4] M. Herda. *The Common Lisp Condition System*. Apress, 2020. doi.org/10.1007/978-1-4842-6134-7
- [5] S.E. Keene. *Object-Oriented Programming in Common Lisp: a Programmer’s Guide to CLOS*. Addison-Wesley, 1989.
- [6] P. Seibel. *Practical Common Lisp*. Apress, Berkeley, CA, USA, 2005. Online version at gigamonkeys.com/book/.
- [7] 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.
- [8] 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
- [9] D. Verna. Interactive and real-time typesetting for demonstration and experimentation: ETAP. *TUGboat* 44(2):242–248, 2023. doi.org/10.47397/tb/44-2/tb137verna-realtime

◊ 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

⁵ tug.org/svn/texlive/trunk/Master/tlpkg/bin/tl-check-files-by-format

Creation of L^AT_EX documents using a cloud-based pipeline

Marei Peischl, Marcel Krüger, Oliver Kopp

Abstract

Using web-based platforms for collaborative editing of L^AT_EX documents is common these days. These tools focus on writing documents and not on creation of templates or packages. Using build servers with automated pipelines is common within software development but can easily be adapted for T_EX & friends.

This article will show how to get started using automated workflows on platforms like GitHub, Forgejo, or GitLab for document authors as well as T_EX developers.

1 Introduction

Everything has become available in or is moving towards the cloud. L^AT_EX is already there for about 10 years and today it's quite common to use a web editor for collaboration and local compilation became the “nerdy” way. But there also is a third variant to compile documents which can be used also to improve package development and in general the stability of L^AT_EX: adapting DevOps methods, like continuous integration and delivery (CI/CD) using automated workflows.

As a very rough working definition, let us consider a CI/CD workflow as a number of steps to compile a T_EX document to PDF on some kind of online service that has access to the source files.

2 Why continuous integration?

Having an established workflow usually makes people avoid thinking about changing anything. So there have to be reasons why it might be worth reading this article, let alone integrating the mechanisms into projects.

Early adopters of continuous integration techniques in the T_EX ecosystems tried to follow the current state of open source development and open doors for contributors in the development process. For example, the L^AT_EX Project is currently welcoming a lot of user interaction via their GitHub projects [16] and also takes contributions from which the whole (L^A)T_EX community will profit.

But the advantages of these methods extend beyond that. We will focus on some cherry-picked aspects as the remainder could probably be an article by itself.

2.1 Works for me?!

Sometimes, I can successfully compile my document on my machine, but my supervisor can't on theirs. There are many reasons why a T_EX compilation may succeed on one system and fail on another. Running some external continuous integration pipeline will not only illustrate the necessary steps to go from source to the full PDF, it will also help understand if problems are machine-specific or general.

2.2 Compatibility and regression testing

With CI/CD, it is possible to run workflows on multiple T_EX distributions or versions. This can be used to test if there are issues with some package update before updating a machine to, e. g., the latest MiK^TE_X updates where downgrading may be complicated.¹

Additionally, one can also use it to check backwards compatibility, e. g., if one collaborator is using a Debian stable version which has only some outdated version. As mentioned, the L^AT_EX Project Team is already using these techniques and even provides functionality for regression testing within their build system, l3build [17, 19].

Using CI/CD as a package developer enables a general interface to be used for regression testing. This allows for avoiding some of the bugs which otherwise would be published and found by a user. Furthermore, it can be used to avoid inconsistent structures, e. g., one of the authors recently found that numerous packages and files within T_EX Live do not have a proper version number set within the code.

3 Structure of this tutorial

The idea is to introduce readers to the basics of setting up automated workflows on GitHub, Forgejo, and GitLab. This tutorial mainly addresses two groups of users:

1. authors focusing on typesetting actual content, including those collaborating on a document, and
2. package or template developers who provide their work to be used by the first group.

The second group obviously can also use workflows of the first, e. g., for typesetting documentation. So developers usually use an extended version of the setup provided for authors.

As all platforms covered by this article are related to the git version control system, we expect the project to be some kind of git repository. In case the reader does not yet use git, there is a little bit of information attached to this tutorial. Using that, it

¹ That's another issue to address ... but a different story.

would be possible to use git without even noticing as it is attached to the autosave function of an editor.

Because the L^AT_EX project is using GitHub, we are going to start with a detailed explanation of GitHub Actions and create a matching setup on the other platforms afterwards. All workflows are available for customization in the template repositories listed in Table 1 near the end of the article. Within this article the listings are marked by icons to not confuse readers as the platform is switched multiple times to show the differences.  is used to indicate the listing belongs to GitHub Actions, whereas  marks the GitLab CI variant.

4 Compiling a document in a CI pipeline

4.1 First steps with GitHub Actions

To get started, we need a git repository somehow hosted on GitHub. It does not have to be public.

Thinking of the tasks needed to compile a L^AT_EX document in any “blank slate” environment, we have to do the following:

1. Prepare the environment and install a L^AT_EX distribution.
2. Run L^AT_EX on the document.

GitHub provides pre-configured actions which are able to combine multiple steps, e.g., the “latex-action” action [20]; this starts another container inside the action container to run `latexmk`. However, these and other actions typically limit the configuration options to simplify the interface. We are going to elaborate on two variants: Using a container image with a full T_EX Live (this section) and a minimal installation (see Section 6).

The configuration for an action is done by creating a yaml file within the repositories’ subdirectory `.github/workflows/hello-world.yml`. The filename may be chosen at will. The following code snippet shows a minimal configuration:

```

1 name: Hello World Action
2 on: [push]
3 jobs:
4   action-test:
5     runs-on: ubuntu-latest
6     steps:
7       - run: echo "Hello world" 🐾 🐱

```

name: of the workflow. This is important if a project contains multiple workflows.

on: This directive indicates conditions under which the workflow will be started. In this configuration, the jobs will be started on any push, i.e., whenever the repository on the server receives an update.

Besides attaching the trigger to some user action it is possible to use time-based settings. For all options it is worth having a look at the configuration manual [5]. The default settings differ for all platforms, and may be specific for a single instance.

jobs: contains a list of jobs to be run one after another. For example, it is quite common to have one job for compiling a document and another one to make the PDF available. In this example, there is only one job called “action-test”.

runs-on: This value corresponds to a runner setup. Runners are the systems that actually execute the jobs defined in a workflow. It does not have to be the same server as the one where the repository is hosted. In this example, “ubuntu-latest” indicates running on one of the provided runners by GitHub which is based on Ubuntu. It includes NodeJS and some tooling to simplify the work using predefined actions. A full list of the readily available runners and detailed description of the images can be found at [4].

steps: This is what the workflow should actually do. As one can see it is possible to directly enter (ba)sh code in there, and use UTF-8. This example merely echoes a string to stdout and therefore should run without any issues.

On GitHub, actions are automatically enabled for new repositories. When combined with an available runner as we do here, it is enough to add a yaml configuration file to the repository to see the effect. After pushing that configuration, the pipeline will start running on all subsequent pushes.

The current status of an action, i.e., what step it is currently running or if it has already finished, can always be checked by having a look at `<repository url>/actions`; e.g., for the first of our demo repositories, this can be found at github.com/islandoftex/tug2024-workflow-github/actions. It looks like this:

✓ Initial Setup 🐕

Build #23: Commit 1d1cfad pushed by TeXhackse ...
 last week  2m 14s [main](#)

The pipeline ran successfully but did not do anything except create some shell output. Hence, we can move on to the next step: building a L^AT_EX document.

4.1.1 GitHub Actions using L^AT_EX

Actions are fundamentally based on isolated containers of software, which are run using Docker. Luckily,

some of us live on the Island of TeX (IoT) and maintain images we can make use of here.² The setup of the images was described within [12].

The first part of the workflow will stay the same for the moment. Changes apply only after **runs-on**:

```
5 runs-on: ubuntu-latest
6 container:
7   image: texlive/texlive:latest
8 steps:
9   - name: Checkout repository
10    uses: actions/checkout@v4
11   - name: Run latexmk
12    run: "latexmk --lualatex"
```

container: Choose the IoT “texlive-latest” image which provides a full TeX Live and some tools [11].

steps: The first step looks different from the one we had before. It’s given a name, “Checkout repository”, which is helpful to simplify debugging as GitHub will tell us which step failed.

uses: is a reference to another action and GitHub’s way to reference pre-configured pipelines encapsulating more complex tasks. In the example, `checkout@v4` refers to a separate repository [6]. This action takes care of the authentication and some internals, so we do not have to deal with those details.

A crucial point is that only after this action are the following steps begun to be executed, within the root directory of our repository.

second step: The second step is also given a name (“Run latexmk”), and then uses the same **run:** directive as our first example, but this time we run `latexmk` [2] with the option `--lualatex` (to use the LuaLaTeX engine, as you might guess). By default, `latexmk` will operate on all `*.tex` files within the root directory. So we do not even have to depend on the file name, as long as we store any files to be included within sub-directories.

4.1.2 Where is the PDF?

If the pipeline succeeds, there will be a green checkmark, but we will fail to find the PDF somewhere. This is because GitHub cannot know which (output) files the user actually wants to see or download. Such files are called “artifacts”, so now we will add another step to the action to keep the PDF and upload it to GitHub as a so-called “artifact”:

² Special thanks to the other islanders at that point!

```
13 - name: Archive documentation
14  uses: actions/upload-artifact@v4
15  with:
16  name: Documentation
17  path: "./*.pdf"
```

After another (successful) run of the pipeline, it is possible to access the PDF wrapped within a `*.zip` at the run’s status page. Clicking on the run on the actions overview page leads there:

Artifacts

Produced during runtime

Name	Size
Documentation	18 KB

Sadly, GitHub currently does not provide an option for individual files without a zip. Also it is not possible to have a static link pointing to the latest artifacts. (Other platforms such as GitLab do provide that feature by default.) To resolve this and make the PDF easier to access on GitHub, it is required to use additional actions. The most common way to do it is publishing the PDF to an orphan branch. Usually this mechanism is used to create web pages and is called “github-pages”.

To be able to write in the repository, it is necessary to adjust the permissions. GitHub provides an interface within the yaml configuration:

```
8 permissions:
9  contents: write
```

This has to be added to the job which should upload the PDF. Additionally, most actions which can be used for uploading artifacts to separate branches request a directory instead of a file. Consequently, all files have to be moved to a separate directory to be used with these actions.

```
15 - name: move pdf
16  run: mkdir -p build && mv *.pdf build/
17 - uses: crazy-max/ghaction-github-pages@v4
18  with:
19    target_branch: pdf-output
20    build_dir: build
21  env:
22    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN
    }}
```

The GitHub token on the last line is required for authentication. Otherwise, the run might be successful, but no PDF will appear on the branch.

4.2 Differences with Forgejo Actions

Forgejo [3] is another code platform which has the aim to be more open than GitHub. It can be self-hosted and provides mechanisms similar to GitHub Actions. They will never act in exactly the same way as GitHub's mechanisms, but are almost compatible.

By default, Forgejo first searches `.forgejo/workflows/` for configuration files. If none are found it will fall back to look inside the `.github/` directory. The only issue you would probably face from simply reusing a GitHub setup is that there are no hosted runners on most instances. Even if there are runners available, they will probably be different from GitHub's.

However, if you can configure a runner yourself and use the same labels as the GitHub runner uses (described in [1]), it is possible to use the same workflow configurations on both. The provided demo repositories on [codeberg.org](#), which is a Forgejo instance, also explain how the runners used there are configured.

4.3 Compiling a document using GitLab CI

GitLab CI is not too different from the previous Actions setup. It is even a bit easier to set up continuous integration here as some steps are run automatically. For example, it is not necessary to check out the repository, because this is done by default.

The configuration file has to be placed within the repository's root directory and given the name `.gitlab-ci.yml`. Here's our same example:

```

1 runlatex:
2   image: registry.gitlab.com/islandoftex/
3     ↳ images/texlive:latest
4   script:
5     - 'latexmk --lualatex'
6   artifacts:
7     paths:
8       - "./*.pdf"
```



Here the steps are simply a list of commands run after each other. Our list just contains one call to `latexmk`.

In contrast to GitHub, GitLab provides an API which allows creating static links to the artifacts. Thus, the `README.md` of the demo repositories there include a link of the following structure:

`<repository url>/-/jobs/artifacts/<branch>/browse?job=<name of job>`

This will list all artifacts attached to the job. For instance, using the example configuration on one of the demo projects results in this sample url:

```
gitlab.com/islandoftex/texmf/tug2024-
workflow-document-gitlab/-/jobs/artifacts/
main/browse?job=check:[latest,lualatex])]
```

5 Testing with multiple versions or compilers

We promised that one can extend these setups to test using multiple `\TeX` versions or engines. Different engines are straightforward to include by running additional steps with different commands. Alternatively, it is possible to use different engines in separate actions or even workflow files. There is no general way to structure those, as the choice always depends on whether to always run everything or save resources via conditional or sequential execution.

To use different versions of `\TeX` Live, the Island of `\TeX` provides historic images of all but the latest `\TeX` Live release (which is not yet historic).

```

1 test-on-IoT-texlive:
2   runs-on: ubuntu-22.04
3   strategy:
4     matrix:
5       image: ["TL2022-historic",
6             ↳ "TL2023-historic", "latest"]
6   name: "Test on ${{ matrix.image }}"
7   container:
8     image: texlive/texlive:${{ matrix.image
9       ↳ }}
9   steps:
```



strategy: this is used to create a loop over elements.

In this case, a variable called `matrix` containing a list of images is used and the content can then be accessed within other parts of the file using `${{ matrix.image }}`. Thus, the first run will use TL 2022, continue on 2023 and finally run on the latest release. A full list of the provided images can be found at [9].

Similarly, GitLab also has a matrix feature. The following example illustrates how to run multiple engines, each on multiple `\TeX` Live releases:

```

1 check:
2   image: registry.gitlab.com/islandoftex/
3     ↳ images/texlive:$TEXLIVE_VERSION
[ ... contains script + artifacts ... ]
4 parallel:
5   matrix:
6     - TEXLIVE_VERSION: ['TL2022-historic',
7       ↳ 'TL2023-historic', 'latest']
8       TEX_ENGINE: ['pdflatex', 'xelatex',
9         ↳ 'lualatex']
```



Here the variable is more like a shell variable but can be used the same way.

6 Minimize the build container

The previous examples used a full TeX Live installation to have the most convenient setup.³ The IoT images even ship all dependencies of additional tools such as arara or minted, which enables that all tools in TeX Live will work out of the box.

Still, sometimes you will not need all the bells and whistles and there are advantages to smaller build containers. For instance, downloading more than one gigabyte can take quite some time. Or you may only have limited disk space available on the runner server.

Side remark: If you are looking to reduce not the images themselves but the compile time because you are building many documents, have a look at last year's IoT article [11].

Back to minimizing the build environment: the most annoying part here might be discovering which packages are actually needed to compile ...

So the Island of TeX proudly presents: DEPP – The DEPENDency Printer for TeX Live [7].⁴

As this article focuses on pipelines we won't go into detail on the tool itself but instead show how DEPP produces a file listing all TeX Live packages necessary for the build. For the example projects these look like:

```
# Proudly generated by the Island of TeX's...
blindtext
cm
:
```

6.1 GitHub

On GitHub there are multiple actions available to install TeX Live as a part of the workflow [13, 18]. Using those, it is possible to minimize the container, which will also reduce the build time.

```
9 - name: Install TeX Live
10   uses: zauguin/install-texlive@v3
11   with:
12     package_file: .github/tl_packages
```

This snippet can be used within **steps:** and makes the **container:** directive obsolete, so it should be removed. The **package_file:** is the path to the DEPP output or a manually created list of packages.

³ Not exactly full: the images we used include neither the documentation nor the source trees of TeX Live.

⁴ If you are German, do not be surprised if this tool is more clever than its name suggests.

6.2 GitLab

On GitLab the simplified syntax makes running a minimized TeX Live a bit more complex. The DEPP repository [7] luckily provides a shell script to install a TeX Live based on the package file. This can be used in the pipelines to modify the container.

```
5   image: registry.gitlab.com/islandoftex...
6   before_script:
7     - minimal_tl_setup.sh "tl_packages"
```

Another option would be to provide a container image which already contains the necessary packages. If a self-hosted runner is used, this is usually the best option as caching can be configured to control when the container is rebuilt or updated.

7 Pipelines for package developers

As promised, the advantages of using automated pipelines are even more significant when used within the development process of packages or templates. In this case, we expect the repository to contain a package and some kind of l3build configuration.⁵

7.1 GitHub

The **latex** step (calling **latexmk** in the examples) is replaced by

```
9 - name: Run l3build
10  run: l3build check --show-log-on-error -q
    ↳ -H
```

This will automatically run all test files according to the l3build setup. As this is within the context of running a package's test suite, the artifacts are totally different. Now, we are not interested in a PDF but rather the test output. One of the present authors has created another action to take care of this [14]:

```
11 - name: Archive failed test output
12   if: ${{ always() }}
13   uses: zauguin/l3build-failure-artifacts@v1
14   with:
15     name: testfiles
16     retention-days: 3
```

Apart from uploading the artifacts, this configuration illustrates another important point: In contrast to the PDF of a document, where we are only interested in the successful output, for test suites we are less interested in success than the failure output. To let GitHub know that we are in fact interested in

⁵ The setup itself also works for other tools, but that is out of scope for this article.

these artifacts, if: `if: ${{ always() }}` has been added. This will also force the step to run even if the previous step failed.

7.2 GitLab

Again, GitLab directly supports the artifact upload without loading external extensions.

```
4   - l3build check --show-log-on-error -q -H
5 artifacts:
6   when: on_failure
7   paths:
8     - ./build/test/*.diff
```



This simplifies the standard setup but is less flexible. For example if you want to use different artifacts for success and failure, it is required to do that within two separate jobs within GitLab.

8 Running locally

During the tutorial at the TUG'24 conference there was a short demonstration of running the pipelines locally. This might be helpful for debugging as one can control the steps manually or to ensure the local setup matches the one on the server.

These setups usually use Docker. For the platforms which use actions there is a tool called “act” to simplify that process using the Docker API.

GitLab extends this with the option to simply have a GitLab runner installed locally. This provides the option of running

```
gitlab-runner exec
```

within a local repository containing some Gitlab CI configuration.

9 This is Continuous Development

Of course every configuration shown here is only an example and can be extended depending on the project’s requirements. It is possible to run arbitrary commands, which might be necessary especially for complex setups.

For example, when creating magazines, it is possible to continuously create a print and an on-line version, excerpts of single articles as well as HTML/EPUB output, while the editors only have to wait for the compilation of the article they are actively working on.

The same goes for study material, where we can have rendered versions including solutions or not, or documents which share links between each other and therefore require many runs.

Integrating these structures into more advanced git usage like a useful branching concept can also help

improve collaboration or simplify the contribution process within open source projects. It will reduce frustration for maintainers as some issues do not have to be checked manually.

10 Conclusion and call for action & feedback

We’ve explained the use of GitHub, Forgejo, and GitLab for compiling L^AT_EX documents and packages. We showed how different releases of T_EX Live and even different compilers can be used to simplify testing across platforms. Also we took care of being able to access the PDF or the testing results in some way. To increase the stability of all parts of T_EX development we hope this will help with more testing of packages and even less waste of time while compiling complex setups.

If you maintain any packages, it would be great if you could try setting up a test to check it against the latest release of T_EX Live or even current TL development. This can also be a preparation for next years T_EX Live pretest, as the Island of T_EX is creating a Docker image for the pretests. If you are planning or attempting to do that and face any issues, we will try to help.

We would love to maintain this as a tutorial to simplify the use of automation for users of T_EX & friends. So if we’ve left open questions, we would love to hear about it and will try to improve this tutorial as well as the examples. Table 1 summarizes the related repositories.

Contributions to this project are very welcome!

References

- [1] Codeberg doc contributors.
docs.codeberg.org/ci/actions/
- [2] J. Collins. latexmk—fully automated L^AT_EX document generation. ctan.org/pkg/latexmk
- [3] Forgejo. Forgejo repository. codeberg.org/forgejo/forgejo
- [4] GitHub. Choosing GitHub-hosted runners. docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#choosing-github-hosted-runners
- [5] GitHub. Workflow syntax for github actions. docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions
- [6] GitHub and contributors. Checkout action. github.com/actions/checkout/
- [7] Island of T_EX. DEPP—dependency printer for T_EX Live. gitlab.com/islandoftex/texmf/depp

Table 1: Template repositories published with this article. The naming scheme is structured as $\langle ci\text{-}type \rangle _ \langle task \rangle$, adding “_minimal” if the example is not using a pre-packaged Docker image but includes methods to install packages based on a dependency file as described in section 6.

All variants listed here have been prepared for at least the three platforms mentioned here (GitHub, GitLab, ForgeJo).

As the urls are quite long, we have published the list including links within the paper’s repository tug.org/l/peischl-cicd2024.

Name	Platforms	Document	l3build	Testing	IoT image
latex	GH GL F	✓			✓
latex_minimal	GH GL F	✓			
latex_testing	GH GL F	✓		✓	✓
latex_testing_minimal	GH GL F	✓		✓	
l3build	GH GL F		✓	✓	✓
l3build_minimal	GH GL F		✓	✓	

- [8] Island of TeX. GitHub workflow template for LATEX packages. github.com/islandoftex/tug2024-workflow-github
 - [9] Island of TeX. GitLab repository: TeX Live Docker image. gitlab.com/islandoftex/images/texlive
 - [10] Island of TeX. GitLab workflow template for LATEX documents. gitlab.com/islandoftex/texmf/tug2024-workflow-document-gitlab
 - [11] Island of TeX. Living in containers—on TeX Live (and ConTEXt) in a Docker setting. *TUGboat* 44(2):249–252, 2023. doi.org/10.47397/tb/44-2/tb137island-docker
 - [12] Island of TeX. Providing Docker images for TeX Live and ConTEXt. *TUGboat* 40(3):231, 2019. tug.org/TUGboat/tb40-3/tb126island-docker.pdf
 - [13] M. Krüger. zauguin/install-texlive repository. github.com/zauguin/install-texlive
 - [14] M. Krüger. zauguin/l3build-failure-artifacts repository. github.com/zauguin/l3build-failure-artifacts
 - [15] M. Peischl, M. Krüger, O. Kopp. Source to this paper, and links to additional resources. tug.org/l/peischl-cicd2024
 - [16] LATEX Project. GitHub organization. github.com/latex3/
 - [17] LATEX Project. l3build—a testing and building system for (L)ATEX. ctan.org/pkg/l3build
 - [18] teatimeguest/setup-texlive-action repository. github.com/teatimeguest/setup-texlive-action
 - [19] J. Wright. l3build: The beginner’s guide. *TUGboat* 43(1):40–43, 2022. doi.org/10.47397/tb/43-1/tb133wright-l3build
 - [20] C. Xu. latex-action. GitHub action to compile LATEX documents. github.com/xu-cheng/latex-action/
- ◊ Marei Peischl
Gneisenaustr. 18
20253 Hamburg
Germany
marei (at) peitex dot de
<https://peitex.de>
 - ◊ Marcel Krüger
Hamburg, Germany
 - ◊ Oliver Kopp
Sindelfingen, Germany
ORCID 0000-0001-6962-4290

A short note on typesetting Latin verse scansion with L^AT_EX and LuaL^AT_EX

Antoine Bossard

Abstract

Large parts of the Latin literature are written in verse: Virgil's *Aeneid* and Ovid's *Metamorphoses* are two well-known examples. As can be noticed from a glance at most specialised textbooks, typesetting the scansion of Latin verses is not trivial. To this end, analysing the syllable quantity, or duration, is required—this is prosody. Conventionally, prosody is denoted with diacritical marks, typically above the vowel of a syllable. Although L^AT_EX provides such a feature, it is not always simple to adjust it to authors' needs. In this short note, several issues arising from the scansion of Latin verses in both L^AT_EX and LuaL^AT_EX are discussed and technical solutions are described.

1 Introduction

Latin verses are based on prosody, that is the syllable quantity, or duration (long, short or common), which in turn defines metrical feet: for example, the foot called dactyl consists of one long syllable followed by two short. The quantity of a syllable is determined by the quantity of its vowel (or vowels, typically in the case of diphthongs).

Scanning a Latin verse is about identifying feet, which requires checking the quantity of all the syllables. To this end, diacritical marks are used: conventionally, a breve (˘) denotes a short vowel, a macron (ˉ) a long one and both diacritics used atop one same vowel indicate that the quantity of the corresponding syllable is common.

A verse is usually split into two with a cæsura, which is also typically indicated at scansion. While feet are conventionally separated by vertical bars (|), the verse cæsura is often materialized with a double vertical bar (||). Elision is sometimes denoted with square brackets or parentheses, and sometimes with a lengthened breve below the space that follows the elided syllable. On the opposite end, a hiatus is sometimes denoted with a lengthened inverted breve below the space that separates the two words forming the hiatus [7].

From various Latin grammar textbooks and dictionaries, it can be deduced that scansion typesetting is a non-trivial issue. For example, the typesetting of diacritical marks is uneven and sometimes lacking in classic textbooks such as [3, 5]. The 2018 edition of [6] is amongst the nicest. More recent textbooks often keep scansion and prosody information at a

minimum [1, 2, 4] and their typesetting sometimes remains below par [8].

In this short note, compiled with L^AT_EX and not LuaL^AT_EX, we review how to typographically realise the scansion of a Latin verse in L^AT_EX and LuaL^AT_EX, and the possible shortcomings of these two solutions. A few technical details are given in an appendix.

2 The case of L^AT_EX

With respect to single diacritical marks applied to single letters, no problem whatsoever has been witnessed when relying on L^AT_EX. In both lower and upper case, both accented Unicode characters (e.g. U+0103 ă, U+0100 Ā) and letters manually marked with the commands \u (breve), \= (macron) (e.g. \u{a}, \=A) are correctly rendered. In the case of manual marking applied to the vowel i, the dotless version of the letter is of course highly desirable: for instance, \=\i instead of \i to obtain ī instead of ī.

Regarding the stacking of several diacritical marks atop one letter, typically to denote a syllable whose quantity is common, it remains difficult with L^AT_EX: \u{\=a} is of no avail as it produces ā. Relying on additional packages is probably the best solution, like tipa by R. Fukui, which notably provides the \u= command to combine a breve above a macron, stackengine by S. B. Segletes or covington originally by M. A. Covington with its \twodias command.

One single syllable sometimes includes several consecutive vowels: diphthongs, like æ, au, eu and œ. The quantity of such syllables is long by principle. So, when denoting prosody, a macron is in most cases expected on top of these vowel combinations. The rendering of æ and œ with either manually set diacritical marks or the corresponding accented Unicode characters is acceptable: ā, Ā, ē, Ē, albeit with the macron being somehow too short. The duration of the æ diphthong can however be short in some words, like *præustus*: ă, Ą are rendered as expected. It is however more difficult to add a diacritical mark above the other diphthongs (i.e. diphthongs for which a ligature is not applicable), such as au and eu, and in the case of synæresis, like in *deest* and at the first syllable of *deinde* (a disyllabic word). Relying on the \overline command is one solution: āū, ēē. The bar this time is slightly too long and is, more or less depending on the font, too close to the letters. Whilst the former issue can be addressed with \mkern: āū, ēē (helpful, by the way, for a lengthened macron too: ā, Ā, ē, Ē), the latter remains.

Finally, it also happens that two consecutive vowels count as a single short one. This is the case, for example, with the ua in *genua*, a disyllabic word.

Antoine Bossard

doi.org/10.47397/tb/45-2/tb140bossard-verse

To avoid complications, publishers often resort to typesetting *genuă*, which can be misleading during scansion. It could be typeset *genvă* for facilitated scansion, but it is not satisfactory either (the letter *v* notably tends to be avoided in favour of the letter *u* in modern Latin text editions). The diacritical mark could also be moved so that it stands between both vowels: *genüa*, but once again we find this solution not optimal (the mark is too narrow). The `\textasciibreve` command combined with `\llap` is not really helpful either: *genüă*. The Comprehensive L^AT_EX Symbol List by S. Pakin does not mention any extensible breve; note that the *stix* package has a `\widecheck` though.

This discussion of a breve over two letters also applies to the stretched breve sometimes used to denote elision: although a `\textasciibreve` lowered below the base line (and negatively kerned) produces acceptable results, like *ego_ipse*, there is still room for improvement. Similarly, the inverted stretched breve to denote a hiatus can be obtained with a lowered and negatively kerned `\newtie`: *modo_ipse*. Acceptable but not quite satisfactory.

3 The case of LuaL^AT_EX

Support for diacritical marks, both single and multiple, strongly depends on the selected font, and this can rapidly become problematic: the desired font may not support them. In addition, a font that works flawlessly with L^AT_EX is not guaranteed to work with LuaL^AT_EX; this is the case, for instance, of T_EX Gyre Pagella (loaded with the package *tgpagella*). This is especially the case when adding diacritical marks on capital letters. Furthermore, this can happen whether Unicode includes the desired character, like U+0232 Y, or not, like V (both letters are incorrectly rendered; they have been rendered separately with LuaL^AT_EX and its default font).

Assuming a font that is aware of such diacritical issues has been selected, LuaL^AT_EX brings significant advantages over L^AT_EX as far as we are concerned (cf. Section 2). First, it enables multiple diacritical marks on a character: ā, ũ (both letters are correctly rendered; they have been rendered separately with LuaL^AT_EX and the Noto Serif font).

Second, the macron over diphthongs can be further improved: not only its width but also the vertical gap between it and the corresponding letters can be adjusted thanks to the `\Umathoverbarvgap` command. The previous *āū*, *ēē* respectively become *āū*, *ēē* (note the increased gap below the macron; both digraphs have been rendered separately with LuaL^AT_EX and its default font).

Next, we acknowledge that Unicode does provide glyphs that could be helpful for our purpose and review their current support with the multiple diacritical mark-aware Noto Serif font. (Note that the results could differ with another font. The support of Unicode by Noto Serif is complete enough so that it is a good candidate for this experiment though.) Once again, the following examples have been rendered separately with LuaL^AT_EX.

First, there is the Combining overline glyph (U+0305), whose purpose is to render a continuous line above several letters. In our case, this could be especially helpful to typeset a macron over diphthongs for which a ligature is not applicable. So, we have applied a Combining overline to each vowel of the *au* diphthong, but the result is not satisfactory: *āū*. In fact, the two overlines are not combined at all.

Second, there is the Combining double macron glyph (U+035E), whose purpose is obvious from its name, and which could once again apply to the macron added to a diphthong scenario. So, we have inserted a Combining double macron between the two letters of the *au* diphthong, but the result is still unsatisfactory: *āū*. Although spanning both letters, the macron is oddly positioned (not centred), and too short.

Third, there exists the Combining double breve glyph (U+035D), whose purpose is to add a breve above two letters. This could apply to two consecutive vowels counting as one single short, as in *genua*. So, we have inserted a Combining double breve between the two letters of the *ua* synæresis, and this time we find the result satisfactory: *ūā*.

Next, it is now possible to rely on the Undertie glyph (U+203F) to typeset a stretched breve to denote elision, which is just fine: *ego_ipse*. Note that in this experiment with the Noto Serif font, the Undertie glyph was unexpectedly “combining” (i.e. like the Combining breve glyph U+0306) and not “spacing” (i.e. like the Breve glyph U+02D8), that is, behaved like the Combining double breve below glyph (U+035C), which seems to contravene the Unicode standard. We thus combined the Undertie glyph with two spaces as a workaround.

Finally, there exists the Inverted undertie glyph (U+2054) which is just fine to denote a hiatus between two words: *modo_ipse*. And just like with the Undertie glyph, in this experiment with the Noto Serif font, the Inverted undertie glyph was unexpectedly “combining” instead of “spacing”; we thus combined it with two spaces to fit our needs.

4 Summary

Regarding L^AT_EX, the support of single diacritical marks applied to a single letter is fully satisfactory. With respect to ligatured diphthongs such as æ and œ, macrons and breves are rendered as expected, although the macron may seem a bit too short. Regarding other diphthongs, like au and eu, and other digraphs induced by synæresis, rather simple commands could do the trick but results can remain imperfect. In fact, in our experiments, while the macron width is all right, it is placed too close to letters and this is not easily adjustable: a completely new user definition is required if such an adjustment is needed. The breve diacritical mark is not extensible, so the result is only average. The most problematic issue arises from double diacritical marks: their support requires an additional package.

Regarding LuaL^AT_EX, Unicode provides useful features for typesetting Latin verse scansion. However, their support strongly depends on fonts, which can provide full, partial (e.g. Noto Serif) or no support at all (e.g. the default LuaL^AT_EX font). For instance, the default LuaL^AT_EX font does not render correctly multiple diacritics nor diacritics on capital letters. For these cases, the Noto Serif font is significantly better.

Finally and on a side note, depending on the font, the \textbar command may produce a vertical bar with too much space around, especially when used within a word: negative kerning will do in such a case.

5 Epilogue

We conclude this note by giving sample Latin verses with their scansion:

ille pedum melior motu fretusque iuventa,
illē pē|dūm mēli|ōr || mō|tū frē|tūsquē jū|vēntā,

hic membris et mole valens; sed tarda trementi
hīc mēm|brīs ēt | mōlē vā|lēns; || sēd | tārdā trē|mēntī

genua labant, vastos quatit aeger anhelitus artus.
gēnūa lā|bānt, vās|tōs || quātūt | āgēr ān|hēlītūs | ārtūs.

multa viri nequiquam inter se vulnera iactant,
mūltā vī|rī nē|quīquām||nītēr sē | vūlnērā | jāctānt,

(Virgil, *Aeneid*, book v, lines 430–433)

References

- [1] P. Boehrer, M.F. Delmas-Massouline, et al. *Bled latin*. Hachette, Paris, 2018. ISBN 978-2-01-170040-7. In French.

- [2] B. Bortolussi. *Bescherelle Grammaire du latin*. Hatier, Paris, 2021. ISBN 978-2-218-93175-8. In French.
- [3] A. Cart, P. Grimal, et al. *Grammaire latine*. Nathan, Paris, 2014. ISBN 978-2-09-171242-0. In French.
- [4] F. Gaffiot. *Le Grand Gaffiot Dictionnaire Latin-Français*. Hachette, Paris, 3rd ed., 2014. ISBN 978-2-01-166765-6. In French.
- [5] R. Morisset, J. Gason, et al. *Précis de grammaire des lettres latines*. Magnard, Paris, 1963. ISBN 978-2-210-47230-3. In French.
- [6] H. Petitmangin. *Grammaire latine*. Nathan, Paris, revised ed., 2018. ISBN 978-2-09-171001-3. In French.
- [7] E. Plantade. Aspects métriques et rythmiques de la couleur archaïque dans les distiques élégiaques d'Apulée. In *Stylistique et poétique de l'épigramme latine : Nouvelles études*, D. Vallat, F. Garambois-Vasquez, eds., pp. 139–153. MOM Éditions, Lyon, 2022. In French. <https://doi.org/10.4000/books.momeditions.16900>
- [8] L. Sausy. *Grammaire latine complète*. Eyrolles, Paris, 8th ed., 2019. ISBN 978-2-212-54685-9. In French.

(Since this article focuses on typography issues, the publication years of the books mentioned in the bibliography correspond to the date of the *dépôt légal* (“registration of copyright”) so as to avoid ambiguity regarding the considered printing.)

A User commands and settings

We give the user-defined commands and settings used hereinbefore. (The author has partially relied on information found on tex.stackexchange.com.)

First, to typeset a breve and a macron over diphthongs, in both L^AT_EX and LuaL^AT_EX:

```
\newcommand*{\dbreve}{[1]}
  {$\backslash$breve$\{$\backslash hbox$\{#1$\}$$\}$\m@th$}
\newcommand*{\dmacron}{[1]}
  {$\backslash$overline$\{$\backslash hbox$\{#1$\}$$\}$\m@th$}
\newcommand*{\dmacronkern}{[1]}
  {$\backslash$mkern 1.5mu
   $\backslash$overline$\{$\backslash mkern-1.5mu$\backslash$ hbox$\{#1$\}%
   $\backslash$mkern-1.5mu$\}$%
   $\backslash$mkern 1.5mu$\backslash$ m@th$}
```

(The `\dmacronkern` command produces a slightly shortened macron, as demonstrated.)

Second, to adjust the vertical gap, for instance to 1.6 pt, between a macron (obtained with `\dmacron` or `\dmacronkern`) and letters, in LuaL^AT_EX:

```
\check@mathfonts
\Umathoverbar{v}{gap}\textstyle=1.6pt
```

(Note that this setting is declared within the main document, that is, not in the preamble, and needs to appear *after* the `\maketitle` command, if called.)

Third, negative kerning, for instance of -1.5 pt , around a vertical bar can be obtained as follows:

```
\kern-1.5pt\textrbar\kern-1.5pt
```

Finally, when denoting elision with a stretched breve below the baseline and there is an elision at the cæsura, we have used a `\makebox` of zero width to typeset a double bar over an undertie:

```
\symbol{"203F}\makebox[0pt]{\textbardbl}
```

This could be considered future work: the stretched breve covers two characters, but three would be better in this case (i.e. a double bar surrounded by spaces).

B Related Unicode glyphs

A summary of the Unicode glyphs mentioned in this note mostly for diacritical marks and ties is given in the table below.

Description	Code point
Breve	U+02D8
Combining overline	U+0305
Combining breve	U+0306
Combining double breve below	U+035C
Combining double breve	U+035D
Combining double macron	U+035E
Undertie	U+203F
Inverted undertie	U+2054

◊ Antoine Bossard
 Kanagawa University, Graduate
 School of Science, Yokohama
 221-8686, Japan
 abossard (at) kanagawa-u.ac.jp
 ORCID 0000-0001-9381-9346

Ac dubitabam tunc an non Pragam essem aditurus.
 Āc dūbī|tābām | tūnc || ān | nōn Prā|gam_ēssem_ādī|tūrūs.

L^AT_EX Tagged PDF project progress report for summer 2024

Frank Mittelbach, Ulrike Fischer

Abstract

The L^AT_EX Tagged PDF project was started in spring 2020 and announced to the T_EX community by the L^AT_EX team at the (online) 2020 TUG conference. This short report describes some of the progress in this multi-year project made during 2024.

Contents

1	Introduction	237
2	Why bother?	237
3	The WTPDF (Well Tagged PDF) examples	238
4	Tagging status of L ^A T _E X packages and classes	238
5	Progress in math tagging	239

1 Introduction

For a description of the background, goals and previous progress reports of the L^AT_EX Tagged PDF project we refer the reader to various previous articles [8, 1, 6, 2]. This report concentrates on a few important additions in the past year.

As a quick summary, the most important goals of the Tagged PDF project are:

- to improve accessibility of PDF documents produced by L^AT_EX;
- to provide tagging of PDF in an automatic and easy way;
- to also improve conversion to HTML;
- to push the use of PDF 2.0—better for accessibility, especially if math is involved;
- and to push improvements in viewers and tools.

On all these topics there has been noticeable progress.

2 Why bother?

In his article about “Signing PDF files” [4], Hans Hagen wrote “*Personally I wonder why one would use PDF to provide adaptive accessibility, because HTML is meant for that.*” This is a sentiment that you encounter quite often: That PDF is such a bad format that it is not worth the time to improve its accessibility.

There is clearly some truth in this. Nobody can deny that HTML is more accessible. It is a structured language, so the structure is there from the start and every viewer that wants to render an HTML page has to understand this structure. Also HTML has a long history of accessibility support, with various

well-known and well-understood standards and implementations. PDF, on the other hand, is a page description language where structure can be added *optionally* (by “tagging” the PDF) and as the structure is not relevant for rendering, its use by viewers is optional too. While there is a PDF standard for accessibility, PDF/UA, it is not really well understood how conforming processors should behave, and it is not easy to test if a PDF is accessible or not. So why do we bother?

The answer is twofold. Firstly, while HTML is a nice format for accessibility it has also some drawbacks. For instance, HTML is not a single file. HTML pages can load graphics, CSS files, JavaScript files, webfonts, session cookies and more from many places and servers. This makes it difficult to store an HTML “document” for offline reading or for archiving. You never know if tomorrow you will see the same thing if you reload the HTML—and so the general advice for keeping evidence about what’s shown on a web page is to make a screenshot or to print into a PDF. The large number of files also makes it difficult to forward or distribute a text in HTML format. In this respect PDF is clearly the better format as it is a single self-contained file and works offline without problems; with PDF/A, a well-known and well-understood standard for archivable PDFs exists.

Furthermore, HTML is not intended to guarantee a faithful and unchanging representation. Instead it deliberately delegates some of the visual presentation decisions to the browser that can adjust the interpretation of the HTML structure to outer circumstances or consumer choices, e.g., the window size, the available fonts, etc. In contrast, PDF provides such a faithful visual presentation of the document, capturing the exact intentions of its author, regardless of the printer or viewer used to render the PDF.

Both models have their important use cases and so it is not surprising that PDF has been widely used for more than thirty years and in all likelihood will continue to be so used when controlled presentation form is important. Thus, PDF is also highly important for users with special accessibility requirement, and the fact that most PDF documents are essentially not accessible is a major concern, which we address with this project.

The second reason is to improve and simplify HTML production. Currently all existing workflows that create HTML from L^AT_EX sources are based on patching and overwriting package code. For example, `tex4ht` contains many `.4ht` files (such as `biblatex.4ht` and `enumitem.4ht`). Analogously, the `latexml` workflow contains many so-called “bindings” (`biblatex.sty.ltxml`, `enumitem.sty.ltxml`),

and the `l warp` package also (`l warp-biblatex.sty`, `l warp-enumitem.sty`). All these files do at their core is essentially the same: they reconfigure L^AT_EX commands and environments so that they produce a structure suitable for HTML. All this configuration work is done without direct contact with the package authors. Thus, if a package changes or extends its features, or if a new package appears on CTAN, all HTML converter have to adapt their configuration files individually and on their own. The Tagged PDF project is different here: while it also changes L^AT_EX commands and environments so that they produce a structure, the goal is to make changes in the kernel and in the packages directly. Once the structure is there, it can then also be used to create HTML without the need of many fragile external patches.

3 The WTPDF (Well Tagged PDF) examples

At the begin of 2024 two standards for PDF 2.0 were finally released: PDF/UA-2, the ISO standard for Universal Accessibility in PDF 2.0 [5] and WTPDF (Well Tagged PDF) for Accessibility and Reuse in PDF 2.0 [10]. The members of the PDF association were asked for examples and the L^AT_EX team was one of the first to provide a reasonably large set of more or less randomly chosen texts. The set covers a variety of document types and demonstrates various tagging techniques—and also open problems. E.g., the Bible, a document with simple tagging but with many structures due to the large number of verses, pushed hard on some limits (it can not be compiled with `pdflatex`) and revealed a number of bugs that slowed down both compilation and validation. The Max and Moritz example demonstrates problems in the handling of documents with more than one language: we were not yet able yet to convince the speech reader to use the correct voice when switching from one language to another. The `amsmath` documentation, `amsldoc`, demonstrates the current state of math tagging.

These examples and many more, along with their sources, can be found at github.com/latex3/tagging-project/discussions/72.

4 Tagging status of L^AT_EX packages and classes

One step of the project is the adaptation of external packages and classes. For this we have created a database that shows the tagging status of various

packages and classes.¹ A week before the TUG conference, the database contained around 200 entries; since then it has grown at great speed and now covers over 1000 packages and classes. The database shows if a package is compatible, partially compatible or currently incompatible with the tagging code and links to issues and test files. If we do not have tests to certify the status it is listed as unknown.

The database is meant as help both for users who want to know if a package can be safely used and for package developers, who can check the status of the packages they maintain.

The database is a `yaml` file. A subset of around 700 entries can be viewed on this web page: <https://github.com/latex-project/tagging-status>.

5 Progress in math tagging

Accessibility of math in a PDF is clearly not very good. In PDF/UA-1 and PDF 1.7 or earlier there is basically no provision for good math tagging—the best one can do is to add some alternative text and hope that the PDF reader doesn't mess up punctuation, etc. In PDF 2.0 there are better options: it supports the MathML namespace and allows to attach additional files (e.g., a MathML representation or the L^AT_EX source of an equation) to the math structure, but these new options are not well supported by consumer applications. The creation and publishing of the WTPDF examples mentioned above triggered some development here. For example, the next release of the Foxit PDF reader [3] will extract a MathML file and pass it on to the AT-technology. Together with changes in the NVDA screen reader [9] it will greatly improve the reading of math. More details can be found in [7]. We expect that other applications will follow, now that there are a growing number of real documents to support.

References

- [1] 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
- [2] U. Fischer, F. Mittelbach. Automated tagging of L^AT_EX documents — what is possible today, in 2023? *TUGboat* 44(2):262–266, 2023. doi.org/10.47397/tb/44-2/tb137fischer-tagging23
- [3] Foxit. PDF reader. www.foxit.com/pdf-reader/
- [4] H. Hagen. Signing PDF files. *TUGboat* 45(1):145–149, 2024. doi.org/10.47397/tb/45-1/tb139hagen-pdfsign
- [5] ISO/FDIS 14289-2; Document management applications — Electronic document file format enhancement for accessibility — Part 2: Use of ISO 32000-2 (PDF/UA-2), 1st ed., 2024. www.iso.org/standard/82278.html
- [6] F. Mittelbach, U. Fischer. The L^AT_EX Tagged PDF project—a status and progress report. *TUGboat* 43(3):268–272, 2022. doi.org/10.47397/tb/43-3/tb135mitt-tagged
- [7] F. Mittelbach, U. Fischer. Enhancing L^AT_EX to automatically produce tagged and accessible PDF. *TUGboat* 45(1):52–59, 2024. doi.org/10.47397/tb/45-1/tb139mitt-deims24
- [8] F. Mittelbach, C. Rowley. L^AT_EX Tagged PDF—a blueprint for a large project. *TUGboat* 41(3):292–298, 2020. doi.org/10.47397/tb/41-3/tb129mitt-tagpdf
- [9] NV Access. NVDA screenreader. www.nvaccess.org/download
- [10] PDF Association. *Well-Tagged PDF (WTPDF)*, Feb. 2024. Version 1.0.0. pdfa.org/wp-content/uploads/2024/02/Well-Tagged-PDF-WTPDF-1.0.pdf

¹ We thank here Ian Thompson who created the initial list and Matthew Bertucci who greatly extended and improved it, providing many test files for future use.

◊ Frank Mittelbach
Mainz, Germany
[https://www.latex-project.org](http://www.latex-project.org)

◊ Ulrike Fischer
Bonn, Germany
[https://www.latex-project.org](http://www.latex-project.org)

LATEX News

Issue 39, June 2024 (LATEX release 2024-06-01)

Contents

Introduction	1	xr: Support links and properties	6
News from the “LATEX Tagged PDF” project	1	Changes to files in the cyrillic category	6
Enhancements to the new mark mechanism	1	Correct definition of \k	6
Providing xtemplate in the format	2	<i>Introduction</i>	
New or improved commands	2	The LATEX Project team remains strongly focused	
doc: Provide \ProvideDocElement	3	on producing automatically tagged PDF output for	
doc: Better support for upquote	3	accessibility and reuse. At the beginning of 2024	
ifthen: Allow active characters in comparisons .	3	the ISO PDF/UA-2 and the WTPDF (well-tagged	
New conditionals: \IfClassAtLeastT and	3	PDF) standards were released and we are glad to be	
friends	3	able to report that it is now possible to use LATEX	
Code improvements	3	to automatically produce documents that conform to these	
Load packages only at the top level	3	new standards. ¹ A sample collection of such documents	
Keep track of lost glyphs	3	ranging from classical texts, such as the Bible, to recent	
Improve fontenc error message	3	technical papers submitted to arXiv.org can be found	
Warn if counter names are problematic	3	at https://github.com/latex3/tagging-project/discussions/72 .	
Extended information in \listfiles	3	In February Ulrike and Frank presented the current	
Optimize creation of simple document	3	project status during the 5th International Workshop	
commands	3	on “Digitization and E-Inclusion in Mathematics and	
Handling of end-of-lines in +v arguments of	4	Science 2024” (DEIMS 2024) at Nihon University,	
\NewDocumentCommand and friends	4	Tokyo, Japan; see [8].	
Declaring appropriate sub-encodings for TS1	4	News from the “LATEX Tagged PDF” project	
symbol fonts	4	In the previous LATEX News [7] we announced some	
Behavior when loading textcomp without options	5	prototype support for tagged tabulars. Some of the	
Rollback improvements	5	necessary code has now been moved from <code>latex-lab</code> to	
Documentation improvements	5	the corresponding packages (using sockets and plugs)	
Further updates to the guides	5	and to the LATEX kernel (for those parts that are also	
Bug fixes	5	necessary for other aspects of tagging).	
Fix inconsistent expansion of the package	5	The kernel code specific to tagging is implemented	
option list	5	in the file <code>ltagging.dtx</code> . For now it contains	
Fix logic for first mark (page region)	5	<code>\UseTaggingSocket</code> , a special invocation command for	
Struts at the end of footnotes or p columns . .	5	sockets that are specific to tagging. This enables us to	
Fix a “missing \item” rollback error	5	also provide <code>\SuspendTagging</code> and <code>\ResumeTagging</code> ,	
Changes to packages in the amsmath category	5	i.e., a very efficient way to temporarily disable the	
amsmath: Correct equation tag placement . . .	5	whole tagging process. This is, for example, necessary	
Changes to packages in the tools category	6	if some code is doing trial typesetting. In that case	
array, longtable, tabularx: Support tagging . . .	6	the trials should not generate tagging structures—only	
array: No \unskip in math cells	6	the finally-chosen version should. Thus, <code>tabularx</code> , for	
verbatim: \verb showed visible spaces	6	example, stops the tagging while doing its trials to	
verbatim: Support tabs in \verb@imput*	6	figure out the correct column widths to use, and then	
multicol: \columnbreak interferes with mark	6	re-enables tagging when the table is finally typeset.	
mechanism	6	Over time, <code>ltagging.dtx</code> will hold more general	
showkeys: Allow \newline in amsthm to work . .	6	tagging code as appropriate. For now it is only	

¹At the present time we are still in a trial/prototype phase in which only a limited set of document classes and packages are supported. Over the next releases we expect to gradually lift these restrictions and eventually provide the full functionality as part of the core distribution, rather than through `latex-lab` modules.

documented as part of `source2e.pdf` but long term we will provide a separate guide for tagging, which will then also include the information currently found in various other places, e.g., `tagpdf.pdf`.

We also added support for a few missing commands described in Leslie Lamport's *L^AT_EX Manual* [1]: If `phase-III` is used the `\marginpar` command will be properly tagged (depending on the PDF version) as an `Aside` or a `Note` structure. In the standard classes `\maketitle` will be tagged if the additional testphase module `title` is used.

The `math` module has been extended and now includes options to attach MathML files to the structures. First tests with a PDF reader and screen reader that support associated files look very promising. Examples of PDF files tagged with the new method can be found at <https://github.com/latex3/tagging-project/discussions/72>.

At last various small bugs and problems reported at <https://github.com/latex3/tagging-project> have been fixed. Such feedback is very valuable, so we hope to see you there and thank you for any contribution, whether it is an issue or a post on a discussion thread.

Enhancements to the new mark mechanism

In June 2022 we introduced a new mark mechanism [2, p. 76] that allows keeping track of multiple independent marks. It also properly supports top marks, something that wasn't reliably possible with L^AT_EX before.

There was, however, one limitation: to retrieve the marks from the page data it was necessary to `\vsplit` that data artificially so that T_EX would produce split marks that the mechanism could then use. Unfortunately, T_EX gets very upset if it finds infinite negative glue (e.g., from `\vss`) within this data. This is not totally surprising because such glue would allow splitting off any amount of material as such glue would hide its size. T_EX therefore responds with an error message if it finds such glue while doing a `\vsplit` operation (and it does so even if a later glue item cancels the infinite glue).

To account for this, the code in 2022 attempted to detect this situation beforehand and if so did not do any splitting but, of course, it would then also not extract any mark information.

In this release the approach has been changed and we always do a `\vsplit` operation and thus always get the right mark data extracted. While it is not possible to avoid upsetting T_EX in case we have infinite negative glue present, it is possible to hide this (more or less) from the user.² With the new code T_EX will neither stop nor show anything on the terminal. What we can't

²A note to l3build users that make use of its testing capabilities: the new mechanism temporarily changes `\interactionmode` and, for implementation reasons in T_EX, that results in extra newlines in the `.log` file, so instead of seeing [1] [2] you will see each on separate lines. This means that test files might show differences of that nature, once the code is active, and must therefore be regenerated as necessary.

do, though, is avoid an error being written to the log file, but to make it clear that this error is harmless and should be ignored we have arranged the code so that the error message, if it is issued, takes the following format:

```
! Infinite glue shrinkage found in box being split.
<argument> Infinite shrink error above ignored !
1. ... }
```

Not perfect (especially the somewhat unmotivated `<argument>`), but you can only do so much when error messages and their texts are hard-wired in the engine.

So why all this? There are two reasons: we do not lose marks in edge cases any more, and perhaps more importantly we are now also reliably able to extract marks from arbitrarily boxed data, something that wasn't possible at all before. This is necessary, for example, to support extended marks in `multicols` environments or extract them from floats, marginpars, etc.

Details about the implementation can be found in `texdoc ltmarks-code` or in the shorter `texdoc ltmarks-doc` (which only describes the general concepts and the command interfaces).

Providing xtemplate in the format

In L^AT_EX News 32, we described the move of one long-term experimental idea into the kernel: the package `xparse`, which was integrated as `lcmd`. With this edition, we move another long-term development idea to stable status: `templates`.

In this context, templates are a mechanism to abstract out various elements of a document (such as "sectioning") in such a way that different implementations can be interchanged, and design decisions can be implemented efficiently and controllably.

In contrast to `lcmd`, which provides a mechanism that many document authors will exploit routinely, templates are a more specialised tool. We anticipate that they will be used by a small number of programmers, providing generic ideas that will then be used within document classes. Most document authors will therefore likely directly encounter templates only rarely. We anticipate though that they will be *using* templates provided by the team or others.

The template system requires three separate ideas

- Template *type*: the "thing" we are using templates for, such as "sectioning" or "enumerated-list"
- A template: a combination of code and keys that can be used to implement a type. Here for example we might have "standard-L^AT_EX-sectioning" as a template for "sectioning"
- One or more *instances*: a specific use case of a template where (some) keys are set to known values. We might for example see "L^AT_EX-section", "L^AT_EX-subsection", etc.

As part of the move from the experimental `xtemplate` to kernel integration, the team have revisited the commands provided. The stable set now comprises

- `\NewTemplateType`
- `\DeclareTemplateInterface`
- `\DeclareTemplateCode`
- `\DeclareTemplateCopy`
- `\EditTemplateDefault`
- `\UseTemplate`
- `\DeclareInstance`
- `\DeclareInstanceCopy`
- `\EditInstance`
- `\UseInstance`
- `\IfInstanceExistsTF` and variants

To support existing package authors, we have released an updated version of `xtemplate` which will work smoothly with the new kernel-level code. The existing commands provided in `xtemplate` will continue to work, but we encourage programmers to move to the set above.

New or improved commands

doc: Provide \ProvideDocElement

In addition to `\NewDocElement` and `\RenewDocElement` we now also offer a `\ProvideDocElement` declaration that does nothing unless the doc element could be declared with `\NewDocElement`. This can be useful if documentation files are processed both individually and combined.

doc: Better support for upquote

In L^AT_EX News 37 [6] we wrote that support for the `upquote` package was added to the `doc` package, but back then this was added only for `\verb`` and the `verbatim` environments. However, in a typical `.dtx` file, most of the code will be in the body of some `macrocode` or `macrocode*` environments, and neither of these was affected by adding `upquote`. We have now updated `doc` so that `upquote` alters the quote characters in these environments as well.

(github issue 1230)

ifthen: Guard against active characters in comparisons

The `\ifthenelse` command now ensures that `<`, `=` and `>` are safe in numeric tests, even if they have been made active (typically by `babel` language shorthands).

(github issue 756)

New conditionals: \IfClassAtLeastT and friends

Around 2020 we added a number of conditionals with CamelCase names, i.e., `\IfClassAtLeastTF`, `\IfClassLoadedTF`, `\IfClassLoadedWithOptionsFF`, `\IfFormatAtLeastTF`, `\IfPackageAtLeastTF`, `\IfPackageLoadedTF`, and `\IfPackageLoadedWithOptionsTF` to help arranging conditional code that depends on the release of a particular class, package or format. However, we only provided the TF commands and not also the T and F variants. This has now been changed.

In 2023 we introduced `\IfFileAtLeastTF` but we did not also provide `\IfFileLoadedTF` at the same

time. This conditional and its T and F variants have now also been added. Remember that one can only test for files that contain a `\ProvidesFile` line. We did the same for the conditionals `\IfLabelExistsTF` and `\IfPropertyExistsTF`, also introduced in 2023.³

(github issues 1222 1262)

Code improvements

Load packages only at the top level

Classes and packages must be loaded only by using the commands `\documentclass` and `\usepackage` or the class interface commands such as `\LoadClass` or `\RequirePackageWithOptions`; moreover, all of these must always be used at the top level, and not inside a group of any type (for example, within an environment). Previously L^AT_EX did not check this, which would often lead to low level errors later on if package declarations were reverted when a group ended. L^AT_EX now checks the group level and an error is thrown if the class or package is loaded in a group.

(github issue 1185)

Keep track of lost glyphs

A while ago we changed the L^AT_EX default value for `\tracinglostchars` from 1 to 2 so that missing glyphs generate at least a warning, but we forgot to make the same change to `\tracingnone`. Thus, when issuing that command L^AT_EX stopped generating warnings about missing glyphs. This has now been corrected.

(github issue 549)

Improve fontenc error message

If the `fontenc` package is asked to load a font encoding for which it doesn't find a suitable `.def` file then it generates an error message indicating that the encoding name might be misspelled. That is, of course, one of the possible causes, but another one is that the installation is missing a necessary support package, e.g., that no support for Cyrillic fonts has been installed. The error message text has therefore been extended to explain the issue more generally.

(github issue 1102)

Warn if counter names are problematic

In the past it was possible to declare, for example, `\newcounter{index}` with the side-effect that this defines `\theindex`, even though L^AT_EX has a `theindex` environment that then got clobbered by the declaration. This has now been changed: if `\the<counter>` is already defined it is not altered, but instead a warning message is displayed.

(github issue 823)

Extended information in \listfiles

The `\listfiles` command provides useful information when finding issues related to variation in package versions. However, this has to date relied on the information in the `\ProvidesPackage` line, or similar:

³By mistake they were initially introduced under the names `\IfLabelExistTF` and `\IfPropertyExistTF`; we corrected that at the same time. This is a breaking change, but the commands have been used so far only in kernel code.

that can be misleading if for example a file has been edited locally. We have now extended `\listfiles` to take an optional argument which can include the MD5 hash and size of each file in the `.log`. Thus for example you can use

```
\listfiles[hashes,sizes]
```

to get both the file sizes and file hashes in the `.log` as well as the standard release information. ([github issue 945](#))

Optimize creation of simple document commands

Creating document commands using declarations such as `\NewDocumentCommand`, etc., provides a very flexible way of grabbing arguments. When the document command only takes simple mandatory arguments, this has to-date added an overhead that could be avoided. We have now refined the internal code path such that “simple” document commands avoid almost any overhead at point-of-use, making the results essentially as efficient as using `\newcommand` for low-level TeX constructs. Note that as `\NewDocumentCommand` makes engine-robust commands, the direct equivalent to `\newcommand` is `\NewExpandableDocumentCommand`. ([github issue 1189](#))

Handling of end-of-lines in +v arguments of `\NewDocumentCommand` and friends

The `+v` argument type provided by declarations such as `\NewDocumentCommand`, etc., allows grabbing of multiple lines of text in a verbatim-like argument. Almost always, the result of this grabbing will be used in a typesetting context. Previously, the end-of-line characters were stored literally as category code 12 (“other”) `^M` tokens. However, these are difficult to work with in general. We have now revised this behavior, such that end-of-line characters are converted to the `\obeyedline` command when parsed by `+v`-type arguments. This change may require adjustments to the source of some documents, but the enhanced ability of users and programmers to exploit the `+v`-type argument means we believe it is necessary.

Declaring appropriate sub-encodings for TS1 symbol fonts

In 2020 we incorporated support for the TS1 symbol encoding directly into the kernel and in this way removed the need to load the `textcomp` package [3] to make commands such as `\texteuro` available.

There is, however, a big problem with this TS1 symbol encoding: only very few fonts provide every glyph that is supposed to be part of TS1. This means that changing font families might result in certain symbols becoming unavailable. This can be a major disaster if, for example, the symbol `\texteuro` (€) or `\textohm` (Ω) no longer gets printed in your document, just because you altered the text font family.

To mitigate this problem, in 2020 we also introduced the declaration `\DeclareEncodingSubset`. This declaration is supposed to be used in font definition files for the TS1 encoding to specify which subset (we have defined 10 common ones) a specific font implements.

If such a declaration is used then missing symbols are automatically taken from a fallback font.

While this is not perfect, it is the best you can do other than painstakingly checking that your document uses only glyphs that the font supports and, if necessary, switching to a different font or avoiding the missing symbols. See also the discussion in [4].

To jumpstart the process we also added declarations to the L^AT_EX kernel for most of the fonts found in TeX Live at the time—with the assumption that such declarations would over time be superseded by declarations in the `.fd` files. Unfortunately, this hasn’t happened yet (or not often) and so many of the initial declarations went stale: several fonts got new glyphs added to them (so their sub-encoding should have been changed but didn’t); others (mainly due to license issues) changed the family name and thus our declarations became useless and the renamed fonts (now without a declaration) ended up in the default sub-encoding that offers only a few glyphs; yet others such as CharisSIL (which triggered the GitHub issue) were simply not around at the time.

We have, therefore, again attempted to provide the (currently) correct declarations, but it is obvious that this is not a workable process. As we do not maintain the fonts we do not have the information that something has changed, and to regularly check the ever growing font support bundles is simply not possible. It is therefore very important that maintainers of font packages not only provide `.fd` files but also add such a declaration to every `TS1...fd` font definition file that they distribute.

To simplify this process, we now provide a simple L^AT_EX file (`checkencodingsubset.tex`) for determining the correct (safe) sub-encoding. If run, it asks for a font family and then outputs its findings, for example, for `AlgolRevived-TLF` you will get:

Testing font family `AlgolRevived-TLF`
(currently `TS1-sub-encoding 9`)

Some glyphs are missing from sub-encoding 8:
`==> \textcelsius (137) is missing`
`==> \texttwosuperior (178) is missing`
`==> \textthreesuperior (179) is missing`
`==> \textonesuperior (185) is missing`

Some glyphs are missing from sub-encoding 7:
`==> \texteuro (191) is missing`

All glyphs between sub-encoding 6 and 7 exist

All glyphs between sub-encoding 5 and 6 exist

All glyphs between sub-encoding 4 and 5 exist

Some glyphs are missing from sub-encoding 3:
`==> \textwon (142) is missing`

All glyphs between sub-encoding 2 and 3 exist

Some glyphs are missing from sub-encoding 1:
`==> \textmho (77) is missing`
`==> \textpertenthousand (152) is missing`

All glyphs between sub-encoding 0 and 1 exist

All glyphs in core exist

TS1 encoding subset for `AlgolRevived-TLF` (ok)

Use sub-encoding 9

This output is meant for human consumption, e.g., you see which glyphs are missing and why a certain sub-encoding is suggested, but it is not that hard to use it in a script and extract the suggested sub-encoding by grepping for the line starting with `Use sub-encoding`.

Of course, this check will only work if the missing glyphs are really missing: some fonts placed “tofu”⁴ into such slots and in this case it looks to `TEX` as if the glyph is provided. For example, for the old Palatino fonts (family `ppl`) it would report

```
TS1 encoding subset for ppl (bad)
Use sub-encoding 0 (not 5)
```

thus it claims that all glyphs are provided, while in reality more than twenty are missing and sub-encoding 5, as declared in the kernel, is in fact correct.

(github issue 1257)

Behavior when loading `textcomp` without options

When incorporating the `textcomp` package into the `LATEX` kernel, in the February 2020 release [3], the default type of its package messages was changed from package info (`Package textcomp Info`) to `LATEX` kernel info (`LaTeX Info`). But if `textcomp` was loaded without options, the message type got restored to package info. This restoration has now been canceled.

Note that loading `textcomp` with one of the options `error`, `warn`, or `info` still changes the message type to an error, warning, or info message from the `textcomp` package.

(github issue 1333)

Rollback improvements

When requesting a rollback of the `LATEX` kernel and/or packages, several packages produced the error “Suspicious rollback date” because their rollback section contained only data about recent releases even if the package, such as `array`, was available since the first release of `LATEX 2 ε` in 1994. We now suppress this error and load the first release that is still part of the distribution (and hope for the best). This change was implemented for the packages `amsmath`, `array`, `doc`, `graphics`, `longtable`, `multicol`, `showkeys`, `textcomp`, and `varioref`.

(github issue 1333)

Documentation improvements

Further updates to the guides

We reported about the updated versions of `usrguide` and `clsguide` in `LATEX News 37` [6]. We have now revised `fntguide` as well to reflect the changes and macros added to the kernel over the last years of development. Note that the file name hasn’t changed and there is no `fntguide-historic`.

⁴Little squares to indicate a missing symbol.

Bug fixes

Fix inconsistent expansion of the package option list

`LATEX` applies one-step expansion to the raw option list of packages and classes, so that constructions such as

```
\def\myoptions{opt1,opt2}
\usepackage[\myoptions]{foo}
```

are supported. But if a package declares its options using the new key/value approach [5] and it gets loaded a second time, then its raw option list will not be expanded and so an error might be raised. This has now been corrected.

(github issue 1298)

Fix logic for first mark (page region)

In the new mark mechanism introduced in June 2022 [5] the result of `\FirstMark` on a two-column page was incorrect if the first column contained no marks. In that case it should have returned the first mark of the second column but didn’t. This has now been corrected.

Documents using `\leftmark` are not affected, because that command is still using the old mechanism for now.

(github issue 1359)

Struts at the end of footnotes or p columns

To produce consistent spacing in footnotes and tabular p-cells `LATEX` adds a strut at the beginning and end of the content. This assumed, however, that the content of the footnote or tabular cell ended in horizontal mode and so, until now, these struts were unconditionally added; as a result, if this content ended with vertical material then this strut started a new paragraph consisting of a single line with just the strut in it. This has finally been corrected and now the placement logic for the strut changes when vertical mode is detected.

(First seen in a bug report for `footmisc` in combination with `bigfoot`)

Fix a “missing \item” rollback error

If `LATEX` is rolled back to a date between 2023/06/01 (inclusive) and 2024/06/01 (exclusive), any list-based environment would raise an error (shown on two lines for `TUGboat`):

`! LaTeX Error:`

`Something's wrong--perhaps a missing \item.`

This has now been corrected as a hotfix in patch level 2, by enhancing the 2023/06/01 version rollback code of the new paragraph mechanism.

(github issue 1386)

Changes to packages in the `amsmath` category

amsmath: Correct equation tag placement

If there is not enough space to place an equation tag on the same line as the equation then `amsmath` calculates a suitable offset and it places the tag above (or below) the equation. In the case of the `gather` environment this offset was not reset at the end, with the result that it also got applied to any following environment, resulting in incorrect spacing in certain situations. This has now been corrected.

(github issue 1289)

Changes to packages in the tools category

array, longtable, tabularx: Support tagging

These three packages have been extended so they can now, on request, produce tagged tabular. This is done by adding a number of sockets (see [7]) that, by default, do nothing; but when tagged PDF is requested they get equipped with appropriate plugs.

In the previous L^AT_EX release this was handled in `latex-lab`, by patching the packages when tagging was requested.

array: No \unskip in math cells

Math cells in the standard `array` environment of the kernel are not subject to space removal at the right end of the cell, i.e., explicit spaces from `\hspace` or `_`, etc. are honored (normal spaces are automatically ignored in math). In the `array` package all spaces got removed by calling `\unskip` unconditionally, regardless of the type of cell. This difference in behavior has now been removed by correcting the processing of math cells in `array`.

(github issue 1323)

verbatim: \verb showed visible spaces

A recent change in the kernel was not reflected in the `verbatim` package, with the result that `\verb` showed visible spaces (`_`) after the package was loaded. This has already been corrected in a hotfix for the November 2023 release.

(github issue 1160)

*verbatim: Support tabs in \verbatiminput**

Mimicking the November 2023 kernel update that allowed `\verb*` to mark tabs as spaces, the `verbatim` package has now been updated so that `\verbatiminput*` also marks tabs as spaces.

(github issue 1245)

multicol: \columnbreak interferes with mark mechanism

The `multicol` package has to keep track of marks (from `\markright` or `\markboth`) as part of its output routine code and can't rely on L^AT_EX handling that automatically. It does so by artificially splitting page data with `\vsplit` to extract the mark data. With the introduction of `\columnbreak` that code failed sometimes, because it was not seeing any mark that followed such a forced column break.

This has now been corrected, but there is further work to do, because as of now `multicol` does not yet handle marks using the new mark mechanism—see the discussion at the beginning of the newsletter.

(github issue 1130)

showkeys: Allow \newline in amsthm to work

Previously `showkeys` added an extra box layer which disabled the `\newline` of `amsthm` theorem styles. This extra box has now been avoided.

(github issue 1123)

xr: Support links and properties

The `xr` package implements a system for eXternal References. The `xr-hyper` package (in the `hyperref` bundle) extended this to also support links to external documents. Using last year's extension of the `\label` command, which unified the label syntax of L^AT_EX and `hyperref`, it became possible to merge the two packages and thus make `xr-hyper` obsolete. With this change it is also possible to refer to properties that are stored in external documents using `\RecordProperties`.

(github issue 1180)

Changes to files in the cyrillic category

Correct definition of \k

Ages ago, the encoding-specific definitions for various accent commands were changed to guard against altering some parameter values non-locally by mistake. For some reason the definition for `\k` in the Cyrillic encodings T2A, T2B, and T2C didn't get this treatment. This oversight has now been corrected.

(github issue 1148)

References

- [1] Leslie Lamport. *L^AT_EX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, 2nd edition, 1994. ISBN 0-201-52983-1. Reprinted with corrections in 1996.
- [2] L^AT_EX Project Team. *L^AT_EX 2_E news 1–39*. June, 2024. <https://latex-project.org/news/latex2e-news/ltnews.pdf>
- [3] L^AT_EX Project Team. *L^AT_EX 2_E news 31*. February, 2020. <https://latex-project.org/news/latex2e-news/ltnews31.pdf>
- [4] L^AT_EX Project Team. *L^AT_EX 2_E news 33*. June 2021. <https://latex-project.org/news/latex2e-news/ltnews33.pdf>
- [5] L^AT_EX Project Team. *L^AT_EX 2_E news 35*. June 2022. <https://latex-project.org/news/latex2e-news/ltnews35.pdf>
- [6] L^AT_EX Project Team. *L^AT_EX 2_E news 37*. June 2023. <https://latex-project.org/news/latex2e-news/ltnews37.pdf>
- [7] L^AT_EX Project Team. *L^AT_EX 2_E news 38*. November 2023. <https://latex-project.org/news/latex2e-news/ltnews38.pdf>
- [8] Frank Mittelbach and Ulrike Fischer. *Enhancing L^AT_EX to automatically produce tagged and accessible PDF*. TUGboat 45:1, 2024. <https://latex-project.org/publications/indexbyyear/2024/>

Web page to PDF conversion with Rmodepdf: Leveraging Lua^LA_TE_X for e-book reader-friendly documents

Michal Hoftich

Abstract

This article presents the use of responsive design methods and advanced features of Lua^LA_TE_X for automatic document typesetting intended for various target outputs, both printed and electronic, such as mobile phones, tablets, or e-readers.

Specifically, it focuses on the use of Lua^LA_TE_X for automated typesetting with the help of the Responsive package [7] for setting font size and line spacing according to page size, the Luavlna package [5] to prevent single-character prepositions at the ends of lines, the Lua-widow-control package [2] to minimize widows and orphans at the ends and beginnings of pages, and the Linebreaker package [4] to prevent line overflow.

1 Introduction

Some time ago, I acquired an e-book reader, but I still read most texts on my PC screen because they come from web sources. It occurred to me that I could save longer articles for later reading on my e-reader. There are, of course, several applications for this purpose, but I decided to create my own, tailored exactly to my needs and preferences. Another motivation is the opportunity to learn something new and create packages that could be useful for other ^TE_X users as well.

My goal is to make the solution as automated as possible, so I don't have to deal with overfull lines or other errors that would require manual intervention. Thanks to the capabilities of Lua^LA_TE_X, such a solution is possible today, as we will demonstrate in the following text.

Because Lua^LA_TE_X provides the Lua programming language in ^TE_X distributions, I used it to create my project Rmodepdf [8]. It uses the LuaxML package [6] to transform HTML into ^TE_X and a few external commands—Curl for downloading of the web pages, and Rdrvview [3], which removes navigation elements, advertisements, and other distractions from the page. Rdrvview is based on the JavaScript library used by the Firefox browser for its Reader Mode. However, it has been translated from JavaScript to C, making it significantly faster and eliminating the need for any additional dependencies.

During the development of the program, I also created or significantly expanded three ^LA_TE_X packages that may be useful on their own. For the LuaxML package, I created an HTML parser that allows

web pages to be processed directly from the Lua language. The Responsive package enables the creation of templates that adjust font size, page margins, and other parameters according to the current page size. Finally, the Linebreaker package prevents line overflow, which is crucial in automated document typesetting where we neither want nor can manually correct such errors.

In addition to saving articles for reading on an e-reader, there are other ways to utilize the Rmodepdf program. One such use is archiving web content on paper. By removing all navigation elements, we obtain a document that can be easily printed, bound, and archived as a book.

2 Basic usage of Rmodepdf

The `rmodepdf` command accepts one or more urls as arguments. It is also possible to use the addresses of local HTML files.

```
$ rmodepdf <url1> <url2>
```

The output file is named based on the title of the first document. If the title cannot be found, a name based on the current date and time is chosen. The generated name is displayed in the terminal output. You can specify a custom file name using the `-o` option.

If you prefer not to compile the document directly but only to display the text generated by the ^LA_TE_X document, you can use the `-p` option.

```
$ rmodepdf -p <url> > foo.tex
```

You can choose a different page size using the `-P` option. By default, the page size and margins are set for e-book readers, but you can also select other sizes, such as A4 paper size. The page style is currently set to empty (*blank*), but you can change it using the `-s` option.

```
$ rmodepdf -P a4paper -s plain <url>
```

For speed, images are stored in a local directory. By default, this is the `img/` subdirectory within the current directory, but you can specify a different directory using the `-i` option.

```
# save the document as foo.pdf and
# save images in the temp dir
$ rmodepdf -o foo.pdf -i /tmp/img <url>
```

You can disable image downloading entirely with the `-n` option. Rmodepdf also detects and displays ^LA_TE_X mathematical commands embedded in web pages that use MathJax or KaTeX for rendering. This default behavior can be disabled using the `-N` option. Additionally, the removal of page elements using Rdrvview can be disabled with the `-R` option.

3 Configuration

3.1 Settings

It often happens that during conversion you encounter errors or wish to change how certain elements on the page are converted into L^AT_EX. Therefore, Rmodepdf provides the option to use a Lua configuration script. This script allows you to modify the code of translated pages, define conversion rules, set variables, or change templates as needed. The configuration file is loaded using the option `-c`.

```
$ rmodepdf -c script.lua <url>
```

The script might look like this, for example:

```
add_to_config {
    document = {
        preamble_extras = [[
            \setmainfont{Linux Libertine O}
        ]],
    },
    img_convert = {
        -- modify the command used for
        -- conversion of SVG images to PDF
        svg = "cairosvg -o ${dest} -",
    },
}
```

Above, the command `add_to_config` is used, which safely copies new configuration values into the original configuration. If you only want to set a single configuration value, you can also directly write to the `config` table:

```
config.document.geometry = "a6paper"
```

The `config` table contains several subtables that you can configure. The `document` subtable includes properties of the output document, such as `preamble_extras` for adding additional code to the document preamble, or `geometry`, which allows you to directly specify the dimensions of the page or margins of the output document.

The subtable `img_convert` defines commands for converting image formats used on the converted web pages that are not supported in LuaL^AT_EX to one of the supported formats. For example, in the sample, we define a command to convert from SVG format to PDF. This command must support reading from standard input, and you can specify the output file name using the template `${dest}`.

The subtable `html_latex` contains settings for translating L^AT_EX code embedded in web pages. The `ignored` item contains a list of HTML elements where embedded L^AT_EX code should not be searched for. Typically, this includes elements like `<pre>`, which contain source code that should not be processed in our document.

The subtable `pages` contains converted files and their metadata. Its content is populated after the configuration script runs, so it is not available beforehand but is utilized in templates. It includes items such as `language` for the document language, `title` for the document title, and `content` which contains the L^AT_EX code of the document for transformation.

3.2 Callbacks

The configuration script is executed before the actual conversion, so it cannot directly influence the conversion process. However, we can define several callback functions that allow us to affect the conversion. These functions are as follows:

```
preprocess_content modify string with the
                    raw HTML before readability and DOM
                    parsing.
preprocess_dom modify the DOM object before
                    fetching of images or handling of MathJax.
postprocess_dom modify the DOM after all
                    processing by Rmodepdf.
postprocess late post-processing of the config
                    table.
```

The most useful are the first three. The `preprocess_content` function takes a string parameter with the HTML code of the page as it was downloaded from the original website, before any modifications by Rdrvew. Here, you can use Lua string functions to fix certain elements that may cause issues during processing with Rdrvew. This method is quite limited and, especially when using regular expressions, it can cause more harm than good. Therefore, use it with caution.

The difference between the next two callbacks is that with the first one, you can still influence image downloading or the processing of L^AT_EX commands. For modifications to the final version of the document, it is best to use `postprocess_dom`.

Both functions receive a LuaXML DOM object as a parameter. This allows you to safely traverse and transform the entire document. LuaXML includes many functions for working with the DOM; here, we will introduce just a few basics. For example, the following example prints the resulting DOM object as HTML code:

```
function postprocess_dom(dom)
    print(dom:serialize())
    return dom
end
```

The `dom:serialize()` method obtains the HTML code from the DOM object, which we then print using the `print` command. It is important to return the DOM at the end of the function; this

ensures that any modifications made to the DOM are preserved and applied to the final document.

Here's a slightly more complex example. Let's assume we want to remove a menu that might look like this, since Rdrvview did not do the removal:

```
<div class="menu">
... menu contents ...
</div>
```

We can use the `postprocess_dom` function to remove this menu:

```
function postprocess_dom(dom)
    -- Find the menu using a CSS selector
    local menu = dom:query_selector(".menu")

    -- Iterate over the menu elements
    -- and remove each one
    for _, el in ipairs(menu) do
        el:remove_node()
    end

    -- Return the modified DOM
    return dom
end
```

In this example:

1. We use the `query_selector` method to find all elements with the class `menu`.
2. Iterate over each element retrieved in the previous step using a `for` loop.
3. Remove each menu element using the `remove_node` method.
4. Return the modified DOM at the end of the function.

This ensures that any remaining menus are removed from the final document.

3.3 Transformation from HTML to L^AT_EX

We perform the conversion of HTML elements to L^AT_EX using the `luaxml-transform` library. This library allows us to declare simple rules for transforming XML or HTML elements into text. Elements can be selected using CSS selectors, which is important because elements with the same name but different classes may need to be converted differently. For example, `` or `<div>` elements are often used as universal tags, but their intended display can vary greatly, depending on their class.

In the configuration file, the `htmlprocess` variable contains an object with rules for converting HTML elements. It provides two main functions: `htmlprocess.reset_actions`, which clears all rules for a given selector, and `htmlprocess.add_action`, which adds new rules. The following code displays some basic usage of the transformation library:

```
htmlprocess.reset_actions("br")
htmlprocess.reset_actions("figure")
htmlprocess.add_action("br", "\n\n")
htmlprocess.add_action("img",
    [[\includegraphics[max width=\textwidth]
        {@{src}}]])
htmlprocess.add_action("figure",
    "\n\n\\medskip\\n\\n\\noindent %s")
```

In this example, we reset the default rules for the `
` and `<figure>` elements and introduce custom rules with specific syntax. The rules adhere to the following conventions:

- The `%s` string inserts the transformed content of the element. It is crucial to include `%s` in most rules to ensure the content is correctly processed; omitting it would hide the entire element's content. However, since the `
` element does not contain any text, it is unnecessary to use `%s` with it.
- In the rule for the `` element, `@{src}` inserts the value of the `src` attribute, which contains the image's address. We use Lua's double-bracket syntax for string constants to avoid C-like interpretation of backslashes.

The following example demonstrates the use of CSS selectors for classes and attribute value comparisons to handle different types of links differently:

```
htmlprocess.reset_actions("a")
htmlprocess.add_action(
    "a.easy-footnote-to-top", "")
htmlprocess.add_action(
    'a[href|="#easy-footnote"]', "%s")
```

Links with the class `a.easy-footnote-to-top` are hidden because the replacement text string is empty. However, for links whose `href` attribute starts with '`#easy-footnote`', only their text content is displayed.

This was just a brief introduction to the transformation possibilities using `luaxml-transform`. You can find many more examples in the LuaXML manual.

4 Template

After converting from HTML to L^AT_EX, we need to combine the resulting code into a single document that can be compiled. Therefore, Rmodepdf includes a simple templating system that allows us to merge individual pages and their metadata together.

A basic template might look like this:

```
\documentclass{article}
\usepackage{linebreaker,responsive}
\usepackage[_{\document.languages}]{babel}
```

```
\usepackage[@{document.geometry}]{geometry}
\pagestyle{@{document.pagestyle}}
@{document.preamble_extras}
\begin{document}
_{pages}
\selectlanguage{@{language}}
?{title}{Title: @{title}}\par{}
?{author}{Author: @{author}}\par{}
\href{@{url}}{@{url}}\par
@{content}
/{\clearpage}
\end{document}
```

Templates contain three syntactic constructs. The basic one is printing a variable using `@{variablename}`. Variables are contained in the `config` table, and using a dot, we can also print properties of subtables. For example, `@{document.preamble_extras}` prints the `config.document.preamble_extras` variable.

The next construct is loops. They have the syntax `_ {variablename}loop code/{separator}`. The variables used have to be arrays. For example, `document.languages` contains the languages of all translated documents in a format suitable for the Babel package, or `pages`, which contains all converted documents. In the loop code, variables of the currently processed array are available. If the array contains only strings, we can use the placeholder `%s`. This is used for `document.languages`. If the current object is a table, we can access its fields directly using `@{variablename}`.

The last construct is conditions. Their syntax is `?{variablename}{true}{false}`. In the example, we use them to insert the title and author, because not all pages have these items.

Custom templates can be read using the `-t` option.

```
$ rmodepdf -t mytemplate.tex <url>
```

5 Automatic typesetting

This brings us to the next part. Since Rmodepdf compiles web pages directly into PDF, we cannot easily intervene in the conversion process. Therefore, the conversion needs to be as automated and error-free as possible. The output PDF can also have various page sizes. The default size is adapted for e-book readers, but we might also want to create a PDF suitable for smartphones or, conversely, a standard A4 size. For all these sizes, we need to choose different font sizes or page margins. This can be achieved using two new packages, which we will demonstrate in this section.

5.1 Responsive design

One of the issues that needs to be addressed is setting the correct font size for readability. The default font size in L^AT_EX is 10 points, regardless of the page size. This is a suitable font size for an A5 page. For A4 format, the font size should be larger, and for smaller screens of e-readers and mobile phones, it can be smaller. Similarly, we can change the line spacing, which also affects text readability depending on the font size and page size.

Web browsers face a similar problem, as they need to display text on large PC monitors as well as on the smaller screens of laptops, tablets, and mobile phones. The solution they use is called *responsive design*.

Responsive design is a way of designing web pages that allows flexible and dynamic adaptation of the appearance and layout of the page content to different display devices. One of the key elements of responsive design is a flexible structure that allows elements on the page to be resized to fit the display device.

Another important element is *media queries*. These allow defining rules that apply based on the properties of the display device, such as screen width and height or the type of output (paper, display). Thanks to these rules, the same page code can be displayed well on both large monitors and mobile devices or when printed.

The Responsive package [7] is inspired by these principles. Its main function is to set the font size according to the page size and the approximate number of characters that should fit on a line. It also sets the typographic scale [9] (affecting font sizes for headings or footnotes), the font baseline, and supports a simple version of media queries.

5.1.1 Setting up the Responsive package

Responsive automatically sets the font size, line spacing, and typographic scale at the beginning of the document. Default values can be changed using package options:

```
\usepackage[<options>]{responsive}
```

or the `\ResponsiveSetup{<options>}` command. The `\ResponsiveSetup` command can also be used directly in the document text, for example, for local font settings changes.

The Responsive package offers the following options:

noautomatic prevents automatic setting of font size and line spacing at the beginning of the document.

characters number of characters for automatic font size setting.

scale typographic scale used for font sizes.

lineratio ratio used in line spacing calculation.

5.1.2 Basic font size

The font size can be set using the `\setsizes` command. Responsive tries to set the font size so that the desired number of characters fits on a line on average. The actual number of characters depends on the text used, as each letter has a different width when using proportional fonts. In practice, the number of characters displayed on a line may be slightly higher.

If the number of characters is not specified in the `\setsizes` command, the value of the **characters** option is used. The following example uses this option setting. Figure 1 shows how the same text can be displayed differently within the same frame, depending on the settings.

```
\begin{minipage}{5cm}
\ResponsiveSetup[characters=55]
\setsizes{}
\lipsum[1]
\end{minipage}
```

5.1.3 Line spacing

By default, L^AT_EX sets the line spacing to the font size multiplied by 1.2. For different fonts and page sizes, different line spacing is appropriate. Similarly, different values may be suitable for the printed and electronic versions of the document.

I was inspired by Edoardo Cavazza's article [1] on readability and added support for setting line spacing based on the ratio of lowercase letter height and the **lineratio** variable. This ratio is obtained by the following calculation:

$$\text{line spacing} = \frac{1\text{ex}}{\text{lineratio}/100}$$

You can observe the impact of changing the **lineratio** value in Figure 2. The choice of its optimal value depends on the font used and the page size. To achieve maximum output readability, it's advisable to compare the output using different values.

5.1.4 Media queries

Media queries are a technique that allows web developers to dynamically adapt the appearance and behavior of web pages based on various device properties, such as screen width and height, device orientation, color support, and many others. With these conditions, it is possible to create responsive and

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

(a) `characters=55`

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

(b) `characters=25, lineratio=38`

Figure 1: Difference in font size depending on the number of characters per line

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

(a) `lineratio=38`

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

(b) `lineratio=34`

Figure 2: Change in line spacing by changing the **lineratio** value

```
 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque.
```

(a) Text width 5cm

```
 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque.
```

(b) Text width 3.9cm

Figure 3: Media query example

flexible web pages that can automatically adjust to different types and sizes of devices on which they are displayed.

How can this technique be useful for L^AT_EX package authors? They could, for example, set the font size, line spacing, and other elements for specific page dimensions. After the user chooses the page size according to the device for which they want to compile the document, these elements are set automatically. The package author can define, for instance, that if the width of the text line is less than a certain size, fewer characters will be displayed on it than on longer lines. The result is shown in Figure 3.

This example will display fewer characters per line if the text width is less than 4 cm.

```
\mediaquery{max-textwidth=4cm}
{\ResponsiveSetup{characters=45}}
{\ResponsiveSetup{characters=60}}
```

A media query can be declared using the `\mediaquery` command, which expects three parameters: the first is a list of tests, the next parameter expects the code to be executed if the tests evaluate to true, and the last one contains the code to be executed if the condition is not met. The code can include the `\ResponsiveSetup` command, as well as any other commands. For example, setting the size of the text block, header, and footer using the `geometry` package.

We can test the following page properties: `paperwidth` and `paperheight` for page dimensions, `textwidth` and `textheight` for text dimensions, `orientation` for text orientation, and `twocolumn` for detecting the use of two-column text in the document.

Tests for text and page dimensions also support the prefixes `max-` and `min-`. Using these, we can test whether a given dimension is smaller or larger than a specified value.

For example, the following command changes the text color to blue if the document has landscape

The example document given below creates two pages by using Lua code alone. You will learn how to access T_EX's boxes and counters from the Lua side, shipout a page into the PDF file, create horizontal and vertical boxes (`hbox` and `vbox`), create new nodes and manipulate the nodes links structure.

(a) Without the Linebreaker package

The example document given below creates two pages by using Lua code alone. You will learn how to access T_EX's boxes and counters from the Lua side, shipout a page into the PDF file, create horizontal and vertical boxes (`hbox` and `vbox`), create new nodes and manipulate the nodes links structure.

(b) With the Linebreaker package

Figure 4: Example of using the Linebreaker package

orientation, the text width is less than 20 cm, and two columns are used.

```
\mediaquery{orientation=landscape,
           max-textwidth=20cm,
           twocolumn=true}
{\color{blue}}
{}
```

5.2 The Linebreaker package

The Linebreaker package [4] prevents text from overflowing in boxes and paragraphs. An example of its output is in Figure 4, where it prevents several lines from overflowing when typeset in a narrow column.

Linebreaker utilizes LuaT_EX's callback which controls line breaking. It replaces the default line breaking function with a modified version that detects overflow or underflow in the broken text. Upon detecting this problem, it retypesets the text with increased values of `\tolerance` and `\emergencystretch` until the overflow is suppressed or the maximum `\tolerance` limit is reached. These changes to `\tolerance` and `\emergencystretch` are local to the currently broken paragraph and do not affect the rest of the text.

5.2.1 Linebreaker configuration

The Linebreaker package can be configured by specifying package options using

```
\usepackage[⟨options⟩]{linebreaker}
```

or later in the document body with the command

```
\linebreakersetup{⟨options⟩}.
```

The options are:

maxcycles the number of attempts to re-typeset a paragraph.

maxemergencystretch the maximum value of `\emergencystretch`.

maxtolerance the maximum value of `\tolerance`.

For example:

```
\linebreakersetup{
    maxtolerance = 90,           % default 8189
    maxemergencystretch = 1em,   % default 3em
    maxcycles = 4,              % default 30
}
```

5.3 Other packages useful for automatic typesetting

We have demonstrated the use of the Responsive and Linebreaker packages for automatic typesetting. These are not the only useful packages that leverage the power of Lua^TE_X for automatic typesetting. Noteworthy examples include Lua-widow-control for suppressing widows and orphans, and Luavlna, which addresses certain typographical issues for Czech and Slovak, while also preventing line breaks in SI units or academic titles.

6 Summary

I hope the demonstration of the Rmodepdf program caught your interest. Even if it didn't, I believe that the side products developed alongside it can be useful on their own.

This includes the capability to process HTML files using the LuaXML package and convert them to ^IA_TE_X using its `luaxml-transform` library. The Responsive package allows you to declaratively set the document design depending on the currently chosen page size. Lastly, the Linebreaker package prevents line overflow, which is a common issue especially in documents with fewer characters per line.

References

- [1] E. Cavazza. Modern CSS techniques to improve legibility. *Smashing Magazine*, 2020. www.smashingmagazine.com/2020/07/css-techniques-legibility/
- [2] M. Chernoff. *The Lua-widow-control package*. Automatically remove widows and orphans from any document. ctan.org/pkg/lua-widow-control
- [3] E. Fernández. *Rdrvview*. github.com/eafer/rdrvview
- [4] M. Hoftich. *The Linebreaker package*. Prevent overflow boxes with Lua^IA_TE_X. ctan.org/pkg/linebreaker
- [5] M. Hoftich. *The Luavlna package*. Prevent line breaks after single letter words, units, or academic titles. ctan.org/pkg/luavlna
- [6] M. Hoftich. *The LuaXML package*. Lua library for reading and serialising XML files. ctan.org/pkg/luaxml
- [7] M. Hoftich. *The Responsive package*. Responsive design methods for ^IA_TE_X. ctan.org/pkg/responsive
- [8] M. Hoftich. *Rmodepdf*. Convert web pages in reader mode to PDF. github.com/michal-h21/rmodepdf
- [9] S. Mortensen. The typographic scale, 2011. spencermortensen.com/articles/typographic-scale/

◇ Michal Hoftich
 Magdalény Rettigové 4
 Praha, 116 39
 Czechia
 michal.h21 (at) gmail dot com
<https://www.kodymirus.cz/>

**Navigating common challenges in manuscript submission:
Insights for authors and publishers using
Elsarticle and CAS packages**

Rishikesan T, Apu V, Rajagopal CV,
Radhakrishnan CV

Abstract

The `elsarticle.cls`, `cas-sc.cls`, and `cas-dc.cls` classes are the preferred L^AT_EX class files recommended by Elsevier, a publishing company specializing in scientific, technical, and medical (STM) content for manuscript submissions to their journals. These class files are available on CTAN along with templates, BibT_EX style files, and user documentations. In this article, we will mainly explore the purpose, features and benefits of these class files and how to utilize these packages effectively for their intended purposes. We also list the challenges resulting from not using these recommended templates.

1 Introduction

As dedicated typesetters with over two and a half decades of experience in using T_EX and related tools for scientific, technical, and medical (STM) journal and book typesetting, and having collaborated with nearly 20 publishers worldwide, we would like to offer some suggestions to authors. These tips can help streamline the publication process and ensure that articles are published quickly, provided they meet all other editorial standards.

Using Elsevier, one of the largest STM publishers, as an example, this advice is applicable to all authors preparing manuscripts for any publisher.

Elsevier, an academic publisher, releases approximately 3,000 journals across various disciplines. These journals follow specific typesetting models characterized by defined margins, text areas, fonts, and column formats (single and double columns). Authors preparing their manuscripts in L^AT_EX for most of these journals are required to use `elsarticle.cls`, `cas-sc.cls`, or `cas-dc.cls`, and format their work according to the guidelines provided on the “Guide for Authors” page. [3, 4]

Given that manuscript publishing is an integral part of research, authors should pay close attention to adhering to publishing guidelines. Compared to the complex nature of their research and the tasks they have already completed with great care and efficiency, manuscript preparation requires only a bit of skill and time, as the data is already available and only formatting remains. The saying “Well begun is half done” certainly applies to manuscript prepa-

ration. If a manuscript is meticulously prepared according to the publishing guidelines and submitted for publication, the remaining process will be much smoother. This careful preparation minimizes the need for additional communications with the publisher, ensuring a hassle-free journey for the authors until their manuscript is published.

2 The purpose

Beyond merely formatting L^AT_EX submissions to a specific style, authors should understand that these class files are designed with several other beneficial purposes to meet the requirements. These additional features ultimately contribute to a faster and smoother publishing process. Before a manuscript is published, it undergoes numerous stages involving human intervention, as well as automatic and semi-automatic processing. Proper use of these class files and following other rules mentioned in the publishing guidelines ensures that each stage is handled efficiently, reducing potential delays and complications. Now let us see the main intended purposes:

1. Maintaining a uniform format for review purposes, especially with wide margins, increased interline spacing, and maintaining the page count if there is any restriction to the number of pages.
2. Assisting authors in formatting equations, tables, textboxes, and other elements to fit within the column and text width of the final published article, to require minimal intervention during the subsequent stages of the publishing process. This careful formatting reduces the risk of errors being introduced in the subsequent stages and streamlines the overall journey of the manuscript through the publishing pipeline.
3. Ensuring that authors include all mandatory information is crucial to avoid rejections. The templates provided with the class files contain most of the required fields, reminding authors to supply the necessary details. If any mandatory information is missing, the publisher will query the authors, causing delays as the information must be submitted later. By providing all mandatory details initially, authors can review the proofs to ensure everything is correctly set and no errors are introduced, again streamlining the publishing process.
4. The templates are meticulously designed for multiple purposes, which we have been discussing here. One of the most important purposes is to automate the typesetting process during publication. When manuscripts are prepared according to these templates, they can be converted into

the required deliverables, such as PDFs, using automated tools. This minimizes manual intervention, significantly reducing the publishing timeline.

5. Manuscripts are published not only as PDFs but also in formats like XML, and MathML; unfortunately, many of the authors are unaware of this fact. When proper templates are used, these conversion processes are largely automated. Otherwise, typesetters must reformat your article to fit these templates, adding time to the publishing timeline. Additionally, proofing tools flag most changes in the manuscript. Authors are then required to review and confirm these changes, increasing their workload and making the proofreading process more challenging.

3 Features

In this section we will discuss the prominent features of these class files. First we will review the features of `elsarticle.cls`. [1, 5, 6, 8]

1. Many features are implemented as class file options: `preprint`, `nopreprintline`, `review`, `twocolumn`, `times`, `sort&compress`, etc. Thus, documents should start with:

```
\documentclass[<options>]{elsarticle}
```

2. Using the `nopreprintline` option, frontmatter which runs on multiple pages can be typeset easily without breaking it manually.
3. The position of the whole front matter can be changed to left-justified, from the default alignment of center, using the `lefttitle` option. This is purely for presentation purpose and is not needed for Elsevier submission.
4. Options to omit loading `natbib`. If any of your packages conflict with the `natbib` package, and you need to use `biblatex` instead, you can do that using the option `nonatbib` as a class option, and then load `biblatex`.
5. Double-blind and single-blind options for peer-reviewing.
6. Structured front matter coding. Authors themselves should code the organisation, city, postcode, state in the respective fields in the template and they are in the best position to identify and format their affiliation correctly. The following is a specimen coding as per `elsarticle.cls`:

```
\begin{frontmatter}
\title{This is a specimen $a_b$}
    \title\tnoteref{t1,t2}}
\tnotetext[t1]{This document is
    the results of the research
    project funded by the National
    Science Foundation.}
\tnotetext[t2]{The second title
    footnote which is a longer
    text matter to fill through
    text width and overflow into
    another line in the footnotes
    area of the first page.}

\author[1]{J.K. Krish\corref{cor1}%
    \fnref{fn1}}
\ead{jkk@example.in}
\cortext[cor1]{Corresponding author}

\author[1,2]{Han Jane\corref{cor2}}
\ead{han@different.edu}
\cortext[cor2]{Corresponding author}

\author[2]{T. Rafeeq\fnref{fn1,fn2}}
\ead[url]{www.nowhere.com}
\fntext[fn1]{This is the first
    author footnote.}
\fntext[fn2]{Yet another author
    footnote.}

\affiliation[1]{%
    organization={Department of
        Physics, J.K. Institute
        of Science},
    addressline={Jawahar Nagar},
    city={Trivandrum},
    % citysep={}, % Uncomment if no
    % comma needed between city and
    % postcode
    postcode={695013},
    state={Kerala},
    country={India}}

\affiliation[2]{%
    organization={World Scientific
        University},
    addressline={Street 29},
    postcode={1011 NX},
    postcodesep={},
    city={Amsterdam},
    country={The Netherlands}}
...
\end{frontmatter}
```

Here is another example, showing frontmatter coding features in the classes `cas-sc.cls` and `cas-dc.cls`. This does not cover all the features, but shows how it is designed. [2, 7]

```
\begin{frontmatter}
\title[mode = title]{Leveraging
social media news}

\tnotemark[1,2]

\tnotetext[1]{This document is
funded by the ABC Organisation}

\tnotetext[2]{The second title
footnote which is a longer text}

\author[1,3]{V. {{\=A}}nand
Rawat}%
[type=editor,
auid=000, bioid=1,
prefix=Sir,
orcid=0000-0001-7500-0000]
\cormark[1]
\fnmark[1]
\ead{a.rawat@txgabcd.org.in}
\ead[url]{www.txgabcd.in}

\credit{Conceptualization of this
study, Methodology, Software}

\affiliation[1]{%
organisation={IMI},
city={Trivandrum},
postcode={695014},
country={India}}


\author[2,4]{Han Xi}%
[style=chinese]

\author[2,3] {Gautham T.}%
[role=Co-ordinator,
suffix=Jr]
\fnmark[2]
\ead{gautham.t@organisation.org}

\credit{Data curation, Writing}

\affiliation[2]{%
organisation={ABC Foundation},
addressline={Jagathy},
city={Trivandrum},
postcode={695014},
country={India}}



```

```
\author[1,3]{Jane Doe}
\cormark[2]
\fnmark[1,3]
\ead{jane@janedoe.org}
\ead[url]{www.janedoe.org}

\affiliation[3]{%
organisation={The Jane Doe
Group},
postcode={MA 12345},
country={USA}}


\cortex[cor1]{Corresponding
author}
\cortex[cor2]{Principal
corresponding author}
\fntext[fn1]{This is the first
author footnote, and is common
to third author as well.}

\nonumnote{This note has no
numbers.}
...
\maketitle
\end{frontmatter}
```

7. Options to insert a page break easily after the title, author/affiliation or abstract. Some journals require, for submission, that authors format the title, or title, author, and affiliation details or the whole frontmatter on separate pages. A page break can be introduced within the frontmatter using the `\newpageafter{title}` command, or the `\newpageafter{author}` command, or lastly the `\newpageafter{abstract}` command. Without these commands, it is difficult to introduce breaks in this area.
8. Environments are defined to code graphical abstract, highlights, etc., properly.
9. Options like `1p`, `3p`, `5p`, `twocolumn`, etc. as per the journal they choose. Even if the author guideline says to submit the manuscript in single column, whereas the final output when the article publishes in the journal is double column, we have advice for authors: first format the manuscript in the final output style (e.g., `5p` which is double column) and format the equations by breaking and aligning the lengthy equations properly. Then submit with `1p` or `3p` option as requested. This will ensure fewer changes during the pre-publishing process.
10. Many `bst` files adhering to both numbered and author-year citation styles are available.

11. In addition to the main title, options are available to code the alternate title, sub-title, translated title and translated sub-alttitle. ORCID links, social media links like X (Twitter), Facebook, LinkedIn, etc., can be given in the optional argument of author command.
12. And many more...

4 The beneficiaries

The primary beneficiaries will be the authors themselves if they follow the template as per the instructions laid out in the guide for authors document.

1. **Authors:** It helps in adhering to the submission guidelines of journals, which may reduce the risk of rejection due to formatting issues. It can potentially speed up the publication process since less time may be needed for formatting revisions during the review stage.
2. **Reviewers:** Consistent formatting across submissions can help reviewers focus on the content rather than getting distracted by varying styles and layouts. Good typesetting ensures that the text is legible, which is particularly important for reviewers who may spend long hours reading multiple manuscripts.
3. **Publishers:** Publishers can expedite the publication process when manuscripts are already formatted correctly, reducing the time to market. Publishers can make revisions and update files as needed without extensive reformatting, maintaining the integrity of the document.
4. **Typesetters:** Using templates reduces the likelihood of errors in formatting, which can save typesetters time in proofreading and corrections. Templates streamline the typesetting process, allowing typesetters to work more efficiently and handle more projects in less time.

5 The challenges

1. The main challenge is authors overlooking the **Guide for Authors**, where the details of the templates with proper instructions are given.
2. For example, authors may merely load the class file, but the rest of the coding will not be according to the template.
3. Not using BIBTEX databases.

6 How to overcome the challenge?

This is a significant question. Based on our experience as typesetters, we suggest the following:

1. Publishers should educate authors by communicating best practices in publishing. Provide numerous use cases and highlight the advantages, including faster publication, of using proper templates. Emphasize the disadvantages of non-compliance with author guidelines. This is a big task!
2. Develop and maintain an effective automatic content profiler that validates author submissions and reports shortcomings back to the authors.
3. Ensure user-friendly documentation is created and made easily accessible.
4. Establish a robust support system with a knowledgeable typesetting team to assist authors and provide timely help.
5. Publishers should actively participate in T_EX conferences and interact with experts to understand the power of T_EX. This will help them modulate policies and communicate effectively with the T_EX author community.
6. Organize free webinars or seminars on best practices for manuscript preparation at universities, especially those with a high number of contributing authors.

References

- [1] ctan.org/pkg/elsarticle
- [2] ctan.org/pkg/els-cas-templates
- [3] elsevier.com/en-in/subject/next/guide-for-authors
- [4] elsevier.com/en-in/researcher/author/policies-and-guidelines/latex-instructions
- [5] support.stmdocs.in/index.php/Elsarticle.cls
- [6] support.stmdocs.in/index.php/FAQ_-_elsarticle.cls
- [7] support.stmdocs.in/index.php/Elsarticle_-_CAS
- [8] support.stmdocs.in/index.php/Model-wise_bibliographic_style_files

◊ Rishikesan T, Apu V, Rajagopal CV,
Radhakrishnan CV
STM Software Engineering Pvt Ltd.,
Trivandrum 695571, Kerala
India
[rishi \(at\) stmsoft dot in](mailto:rishi(at)stmsoft(dot)in),
[apu.v \(at\) stmsoft dot in](mailto:apu.v(at)stmsoft(dot)in),
[cvr3 \(at\) stmsoft dot in](mailto:cvr3(at)stmsoft(dot)in),
[cvr \(at\) stmsoft dot in](mailto:cvr(at)stmsoft(dot)in)
<https://stmsoft.org>

Making BachoTeX proceedings—extended version*

Jean-Michel HUFFLEN

Abstract

We report an experiment of making the proceedings of a conference by automatically generating as much information as possible. For example, we look for titles and other metadata in successive source files to build the table of contents. Separate source files must be processed by a \TeX -like format, which may be pdf \TeX , X \TeX , Lua \TeX or ConTeXt. Installing our functions requires a UNIX-like `make` command and some programs written mostly using Scheme, partly using Ruby.

0 Introduction

We expose the method we used to make the 3-year proceedings of the Polish-speaking \TeX user conference (BachoTeX). These proceedings have some specific aspects, but we think that our method may be applied—even improved—to similar contexts. Our guiding idea: the process of grouping successive articles in one printable file should be automated as far as possible. Besides, we succeeded in avoiding *information redundancy*. For example, the metadata for successive articles—authors, titles and page ranges—are *extracted* from successive source files and the two tables of contents are automatically generated.

In the first section, we introduce to our framework's guidelines and show how our programs behave, we explain why we gained valuable experience through previous projects and why we have volunteered for this task, which should be finished before the summer's end. Section 2 gives the features specific to BachoTeX conferences. Then the successive steps of our *modus operandi* are described in Section 3. Some paths to go further and improve our tools are discussed in Section 4. We give some examples of short programs throughout this article, but reading them requires only basic knowledge in programming.

1 Genesis: Cahiers GUTenberg, #58

In the fall of 2020, there was some substantial reorganisation of GUTenberg, the French-speaking \TeX users group.¹ This process and its reasons have already been described and discussed in many places,

* A first version of this article [15] has been included in the proceedings of BachoTeX 2024.

¹ In French: *Groupe francophone des Utilisateurs de \TeX*.

including [1, 16], so hereafter we do not give more details about that. We just mention that after a period of lethargy, a new board of directors was elected. During this new board's first meeting, we were appointed as the new editor of this group's journal: the *Cahiers GUTenberg*, as already reported in [13].

When we started our mandate, in the beginning of 2021, we inherited many article proposals, more or less recent, more or less ready. Some used $\text{\LaTeX}_2\epsilon$ packages incompatible with other engines, some showed Lua \TeX [11] abilities, some were compilable only by X \TeX ² [28]. In other words, we had to use *several* typesetting systems for composing the same issue. Besides, putting *Cahiers GUTenberg*'s articles on the Web after a while—exactly like *TUGboat*'s—was planned. That is, for an issue, the whole of it will be available, as well as separate articles, including editorials. In addition, let us mention that some .pdf³ files resulting from such articles may be *interactive* or include *animation* effects—e.g., by means of the *animate* package [10]—when read on line. More precisely, each separate article could be given *printed* and *online* versions—the latter using *hyperlinks* by means of the *hyperref* package [27]—, additional versions being available if they are of interest. The printed version of the whole issue is easily built by means of the *pdffpages* package [20].

Controlling the whole process by means of *check programs* may seem perfectionist, or be viewed as a mind game. However, we have personally experienced a situation in which such a control program would have been very useful: the French translation of *The \TeX Companion*'s second edition [22]. As time began to run out, a last change shortened a chapter from two pages: the body's page numbers were updated, but not the page numbers put within the index.⁴ We understood that lesson: as soon as we were in charge of the *Cahiers GUTenberg*, we decided to implement programs to automate this process as far as possible—e.g., building the table of contents—other programs are devoted to some checks—the succession of page ranges within the editorial and subsequent articles.

Implementing all our programs by means of additional $\text{\LaTeX}_2\epsilon$ classes or additional options of our

² We were told that a source file compiling with X \TeX should compile with Lua \TeX . But at this time, that was not always the case; we were forced to acknowledge that as evidence. We asked many people about that; no one was able to explain why to us.

³ Portable Document Format.

⁴ This mistake has been pointed out in [26], even if the book's overall review is very good.

```

(define-record-type <cg-contents-info-record> ; Defining a structure's components for keeping the suitable
...)) ; information.

(define cg-current-contents-info ; Defining a placeholder for the information to be retained when a .tex file
  '*dummy-value*) ; is parsed. Processing a new file causes this variable to be reinitialised.

(define cg-toc-function
  (g-mk-tex-parsing-f
    (g-retain-command "author" 1 #f ; No optional argument.
      #t ; Top level only.
      #f ; Recursive search if files are input.
      #t ; Preamble only.
      #t ; Get all the occurrences.
      (lambda (author-s) ; Function to be applied as soon as this command is to be
        ...)) ; processed.

    (g-retain-command "title" 2 #t ; The first argument is optional.
      #t #f #t 1 ; Only one occurrence, then give up.
      (lambda (ignored-s title-s)
        (cg-contents-info-set-title! current-contents-info
          (normalize-space title-s))))))

...)

;; What is launched: processing all the main source files, in turn.
(writeln/return (cg-make-toc cg-source-jobname-list ; List of main source file pathnames.
  "issue-58-toc-contents.tex")) ; Pathname for the result.

```

Figure 1: How the table of contents of *Cahiers GUTenberg* #58 has been built.

class for the *Cahiers GUTenberg* would have been possible,⁵ but we think that this is *misusing* TeX, because these additional tasks are related to ‘*actual*’ programming more than *typesetting*. As we explained in [13], we think that TeX should not be viewed as a universal multi-task program. TeX’s language has the same expressive power as a Turing machine, so theoretically any function can be programmed using TeX’s primitives,⁶ but as with any specialised language, using it for a purpose other than its intended one may be tedious. The program described in [13], written using the Scheme programming language,⁷ aims to do this kind of task: it allows a (L)ATeX source file to be parsed, with particular focus on commands of interest (the rest is just skipped). As an example, extracting the accurate information in order to build the successive items of the table of contents of [2] is sketched in Fig. 1. More details about configuring our (L)ATeX parser in Scheme can be found in [13]. Going back to building a table of contents’ successive items, the title is of course given by means of the \title command:

```
\title{\foreignlanguage{english}{Donald
  Knuth} : des mathématiques à la
  typographie}
```

The cedram-cg-8 class, designed for *Cahiers GUTenberg*, uses LATEX’s predefined command \author, but *each* author among several co-authors must be specified by a separate command:

```
\author{\firstname{Sandrine}
  \lastname{Chenevez}}
\author{\firstname{...} \lastname{...}}
...
```

Of course, getting the first and last names of an author is a bit more complicated than getting a title, considered as a whole. We do not detail our method in Fig. 1, but in reality, it is quite long but not very difficult, either. Likewise, getting the first page number is easy. If we are interested in checking the succession of page ranges, the last page number of an article can be got by means of the lastpage package [23], which puts a new label *LastPage*: our program is able to parse an auxiliary (.aux) file to catch the information associated with it.

We have just sketched *what* we are doing, but not *when*. Gathering several articles does not mean that they have been finalised w.r.t. the order of appearance; some last minute change may occur, and in some cases, performing all the check procedures

⁵ Besides, an option of the previous version of the class used for the *Cahiers GUTenberg* allowed metadata for Web search engines to be generated. The present version does not.

⁶ Interested readers can consult [3] about this subject.

⁷ A good introductory book to this functional programming language is [30].

Setters:	<code>\hbsettitleinenglish</code> <code>\hbsettitleinpolish</code>	specify a title's versions	analogous to L ^A T _E X's predefined command <code>\title</code>
Getters:	<code>\hbgettitleinenglish</code> <code>\hbgettitleinpolish</code>	place a version of the title within a text the internal command <code>\@title</code>

Table 1: Getters and setters for a title.

again might not be needed. In other words, a program able to determine which procedures are to be run again or have not been run yet is welcome. Historically, the UNIX system's `make` command was the first to automate a sequence of tasks, launching them only if need be [8]. Roughly speaking, a command controlled by `make` is launched if one of its *sources* is more recent than its *target*: in other words, such a command is run if its target does not exist or must be built again. The most frequent case is the production of an executable file from source files. The information concerning sources and targets of such commands is given in a configuration file so-called *Makefile*:

```
target: source1 source2 ...
       command
```

Such a command is useful in software development, especially if a *huge* number of source files are handled. In truth, these configuration files' syntax may be considered old-fashioned and more recent programs, more user-friendly, have appeared, but they are devoted to particular programming languages, most often object-oriented. A 'make-like' command, specialised for engines and formats built out of T_EX is `latexmk` [7]. However, we did not use it because it does not work with ConT_EXt; it does not allow a Scheme program to be launched, either.

In addition, let us mention that according to `make`'s terminology, a target may be a generated file's name—e.g., the result of compiling a source file—or just denote a set of actions. 'Traditionally', some particular targets are used to remove useless files, as soon as the installation is performed (`make clean`) or to revert to the state at the installation's beginning (`make distclean`). So such tasks can be integrated into our framework.

When we ended up with these programs, the company in charge of printing and binding this first issue [2] was given:

- the contents (an additional final page was for the company's brand), using the A4 paper size with cropmarks;⁸
- the complete cover, including the spine, rightly dimensioned;

⁸ *Cahiers GUTenberg*'s first issues used A4. More recent issues' paper size is 165 mm × 225 mm.

as two .pdf files. Generating online or additional versions of separate articles is done by Ruby programs [14], based on *regular expressions*.⁹

2 Exodus

In 2018, 2019 and 2023, BachoT_EX conferences, organised by the Polish GUST¹⁰ group, did not provide proceedings, unlike what was practised before. Given our experience (cf. §1), we volunteered for this task at BachoT_EX 2023's end. Here are the differences, in comparison to *Cahiers GUTenberg* #58:

- we planned a 3-part volume, each part for a conference;
- *two* tables of contents are to be put: a Polish one at the beginning, an English one at the end;
- some authors provide *two* abstracts, in English and Polish: we personally wish to homogenise the layout of such 2-language abstracts;
- authors writing in Polish use different methods for that: the packages `babel` [4] or `polyglossia` [6], or an *ad hoc* package `polski` [24]; as a consequence, the definitions applied to fragments written in Polish should be adaptable;
- here also, typesetting systems depend on articles: an article showing graphical features was based on the PostScript language, other articles use ConT_EXt [12].

We have considered that processing ConT_EXt source texts is a worthwhile exercise, allowing us to be able to accept such texts for the *Cahiers GUTenberg* in the future. Let us also notice that processing the whole of proceedings by means of ConT_EXt is possible, but only if all the articles have been typeset with ConT_EXt.

When an article for the *Cahiers GUTenberg* is written in French, the `cedram-cg-8` class provides two commands for English and German versions of the title, but only the title in French is included into the table of contents. On the contrary, the articles

⁹ Regular expressions can be viewed as an advanced language for describing *patterns*—a good introductory book is [9]—: they are implemented in many programming languages, including Ruby [19]. However, they have not been included into the present standard of Scheme [29].

¹⁰ *Polska Grupa Użytkowników Systemu T_EX*, that is, the Polish-speaking T_EX user group.

of BachoTeX conferences may be in English or Polish. The commands we introduce for titles are given in Table 1, they are grouped into *setters* and *getters*, using object-oriented terminology. Such new commands are prefixed by ‘hb’ and have been put within a package so-called `hbachotex`. The following example illustrates how these commands can be used:

```
\hbsettitleinenglish{%
  Making an Issue of the \cgut}
\hbsettitleinpolish{Skład biuletynu \cgut}
\title{\hbgettitleinenglish\thanks{%
  Title in Polish:
  \foreignlanguage{polish}{%
    \emph{\hbgettitleinpolish}}.}}
```

and it shows that the `\cgut` command should be known when a table of contents is processed, that is, if such a title—in English or Polish—is processed by L^AT_EX again. That has been solved by the following convention within the source `.tex` file:

```
%%%--smlbibtex-plus--[
\newcommand{\cgut}{%
  \foreignlanguage{french}{%
    \emph{Cahiers GUTenberg}}}
%%%--]
```

`\hbsettitleinenglish{...}`

When a output file is open by our Scheme program, all the fragments surrounded by a starting line:

`%%%--smlbibtex-plus--[`

and an ending line:

`%%%--]`

can be inserted into this output file.

3 Leviticus

We have given an overview of the programs we developed and sketched how they have been written. Now we describe how successive stages are chained. Of course, some errors may cause backtracking within the complete process.

3.1 Creating a master file

First you have to write a *master file*, as sketched in Fig. 2. It is a L^AT_EX source file, but the most important decision, at this step, concerns the articles’ succession, as shown in Fig. 2. Such a master file should be located at the root of the directory containing subdirectories for articles; it also loads the `hbachotex` package, but with a `complete` option, which gets access to more commands used for finalisation.

When such a file is processed by L^AT_EX, let us notice that some arguments are *useless*: for ex-

```
\documentclass[onecolumn,a4paper]{ltugproc}
...
\setcounter{page}{3}
\begin{document}
\hbinputtoc{polish}{polish-toc}
\hbincludearticleplus{%
  pdflatex}{editorial}{editorial}
\hbpart{BachoTeX 2018}{}
\hbincludearticleplus{%
  xelatex}{B-2018/hufflen}{hb-greg}
...
\hbpart{BachoTeX 2023}{}
...
\hbincludearticleplus{%
  lualatex}{B-2023/bolek/foto}{foto}
\hbincludearticleplus{context}{B-2023/egger}{%
  Pocketdiary-modul-2023-tugboatstyle}
...
\hbinputtoc{english}{english-toc}
\end{document}
```

Figure 2: Successive articles for BachoTeX conferences in 2018, 2019, and 2023.

ample, the first arguments of the two commands `\hbinputtoc` and `\hbincludearticleplus`:¹¹ the former is the language of the table of contents, the latter is the typesetting system to be used.¹² These commands themselves will just process a source file or insert a `.pdf` file, but the information given by its first arguments will be caught by a Scheme program and used to generate tables of contents or separate articles.

¹¹ Concerning the `\hbinputtoc` command, its second argument is the file name for the corresponding table of contents. The last two arguments of the `\hbincludearticleplus` command are the directory where source files are located and the job name.

¹² In reality, this `\hbincludearticleplus` command’s first argument may be any executable program, provided that it’s available in your *path*.

```

polish-toc ...: $(all-source-files)
    $(SCHEME) $(CHECK_MAKE_TOCS) $(all-source-files)

editorial/editorial.pdf: editorial/editorial.tex
    cd editorial ; pdflatex editorial.tex

B-2018/hufflen/hb-greg.pdf: B-2018/hufflen/hb-greg.tex
    cd B-2018/hufflen ; xelatex hb-greg.tex

...

```

Figure 3: Generation of dependencies.

3.2 Playing with a skeleton Makefile

1. Our toolbox provides a skeleton file you can use to create your `Makefile`, in which only basic targets—e.g., `clean`, `clean-deps`, `realclean`, `distclean`, ... — and general definitions:

```
SCHEME = ...
```

are specified. Within this configuration file, you can notice an input order for another configuration file, `deps`, not yet existing:

```
+include deps
```

(the `deps` file's contents is to be inserted, but ‘+’ prevents errors if this file does not exist);

2. Run the program `hb-makedepend.scm`—or, better, the `deps` target:

```
make deps
```

—to compute the dependencies from the master file's `\hbincludetitleplus` commands: the result is the `deps` file. Fig. 3 sketches the result corresponding to Fig. 2's example.

3.3 Generating .pdf files

1. Now you can generate the separate articles by means of suitable targets, as soon as you make precise starting page numbers.
2. An extended version of Fig. 1's program:

```
hb-maketoc-plus.scm
```

allows all the page ranges of successive parts and articles to be checked. Then the two tables of contents are generated as files input within our master file.

3. The complete file can be built, either by a suitable target of the `Makefile` configuration file:

```
make all
```

or by compiling your master file with pdfL^AT_EX.

3.4 Post-process

Some targets allow a cover to be built, possibly with a spine, correctly dimensioned. Then you can delete files that have become unnecessary. You could build online versions of separate articles, as we did for the *Cahiers GUTenberg*.

4 Numbers

4.1 Requirements

To use our toolbox, you will need, in addition to a working distribution of L^AT_EX & Co:

- a Scheme compiler (preferred) or interpreter, R7RS-compliant [29]: a good example is Racket [18], but other programs exist;
- a make utility recognising GNU¹³ features [31];
- a Ruby interpreter if you are interested in generating variants.

4.2 Discussion

When we work on an article to be included into the *Cahiers GUTenberg* or BachoT_EX proceedings, we put or check commands for titles and specify the first page number. Sometimes we also check or adapt multi-language abstracts. Of course, we make sure that an article's source files are compiled correctly;¹⁴ we also allow ourselves to make some slight adjustment if we can avoid errors or warning issued by typesetting systems. At this step, we deal with bibliographies: sometimes a `.bb1` file is provided, sometimes we have to run BIBT_EX [25] or biber [17].

The dependency relations we generate, as shown in Fig. 3, do not include the production of bibliographies. They do not consider the source files handled by the `\input` command or the graphic files processed by the `graphicx` package's `\includegraphics` command [5]. Roughly speaking, our dependency relations are sufficient for the operations we perform, but should not be used to make ready articles unfinished.

4.3 Going further

When we adapted our toolbox to the BachoT_EX proceedings, the most time-consuming operation was the implementation of a new module for ConT_EXt, so-called `s-tugboat-hbachotex.mkx1`, more or less equivalent to the `ltugproc` L^AT_EX class. In other contexts, the `\author` command's taxonomy depends

¹³ Recursive acronym: GNU's Not UNIX.

¹⁴ We also ask authors to provide the `.pdf` file they got with their installation, if possible.

on classes, as mentioned in §2. However, we would be confident if we had to apply our method in other analogous situations.

In future versions, we plan to study the use of the `latexmk` command. Of course, we will have to propose an alternative for source texts processed with ConTeXt. We could also be able to automate the generation of *metadata* using the `.bib` format for complete proceedings, as well as for separate articles.

5 Conclusion

After some last minor details, we expect *Cahiers GUTenberg* #59 and BachoTeX proceedings to be ready before fall 2024. The future will confirm that or not. Then our programs will be hosted at the servers of the French and Polish groups, as agreed. We think that our system is *open*, because some typesetting systems can coexist, some natural languages, too. We recognise that it is limited because some operations require some experience in programming, especially in Scheme. But we hope that some additional experience will allow us to provide turnkey programs.

Acknowledgements

I thank Karl Berry and Barbara Beeton, who proofread this article. Last but not least, Hans Hagen helped us a lot about aspects related to ConTeXt.

A Additional packages

In this appendix, we briefly mention two packages included in our toolbox.

A.1 The `antispam` package

This first package allows the definition of commands changing the occurrences of the characters ‘@’ and ‘.’ within an electronic address. Most often these two characters are substituted respectively by ‘(at)’ and ‘(dot)’, in which case the `\antispamemail` command of this package may be used. Otherwise, you can put other conventions into action. Notice that if the electronic address is parsed by another command C , we have to define a *robust* command in order for the electronic address to be pre-processed before C is applied. For example, we introduced our electronic address as follows at this article’s beginning:

```
\asnetaddress{jmhuffle@femto-st.fr}
```

the `\asnetaddress` command being defined by:

```
\DeclareRobustCommand{\asnetaddress}[1]{%
\netaddress{\antispamemail{#1}}}
```

A.2 The `more-abstracts` package

This second package allows the definition of additional environments for *several abstracts*, in succession and written using different languages. The `abstract` environment is for the English language; other environments are provided for Polish, Spanish, Portuguese. Creating new environments for other languages is possible, too.

References

- [1] Jacques ANDRÉ, Patrick BIDEAULT, Denis BITOUZÉ, Thierry BOUCHE, Michel BOVANI, Maxime CHUPIN, Daniel FLIPO and Yvon HENEL: “The Last Decade at GUTenberg”. *TUGboat*, vol. 43, no. 1, pp. 7–9. 10.47397/tb/43-1/tb133andre-gut. 2022.
- [2] ASSOCIATION GUTENBERG : *Ils sont de retour!*, Vol. 58 de *Cahiers GUTenberg*. 10.60028/cahiers.v2021i58. Septembre 2021.
- [3] Pieter BELMANS: *A Turing Machine in LATEX: the Follow-Up*. 12 December 2010. Universiteit Antwerpen. pbelmans.ncag.info/blog/2010/12/12/a-turing-machine-in-latex-follow-u.
- [4] Javier BEZOS and Johannes L. BRAAMS: *babel. User Guide. Version 24.7*. 26 June 2024. mirror.ctan.org/macros/latex/required/babel/base/babel.pdf.
- [5] David L. CARLISLE and THE LATEX PROJECT: *Packages in the ‘Graphics’ Bundle*. 22 May 2024. [mirror.ctan.org/macros/graphics/grfguide.pdf](http://mirror.ctan.org/macros/latex/required/graphics/grfguide.pdf).
- [6] François CHARETTE, Arthur REUTENAUER, Bastien ROUCARIÈS and Jürgen SPITZMÜLLER: *polyglossia: Modern Multilingual Typesetting with XE \LaTeX and Lua \LaTeX* . 15 July 2024. mirror.ctan.org/macros/unicodetex/latex/polyglossia/polyglossia.pdf.
- [7] John COLLINS, Evan MCLEAN and David J. MUSTLINER: *latexmk — Fully Automated LATEX Document Generation*. 7 April 2024. ctan.org/pkg/latexmk.
- [8] Stuart I. FELDMAN: “*Make — A Program for Maintaining Computer Programs*”. *Software: Practice and Experience*, vol. 9, no. 1, pp. 255–265. April 1979.
- [9] Jeffrey E. F. FRIEDL: *Mastering Regular Expressions*. 3rd edition. O’Reilly. August 2006.
- [10] Alexander GRAHN: *The animate Package*. 18 June 2023. gitlab.com/agrahn/animate.

- [11] Hans HAGEN: “LuaTeX: Howling to the Moon”. *Buletyn Polskiej Grupy Użytkowników Systemu TeX*, vol. 23, pp. 63–68. April 2006.
- [12] Tom HOTTEN and Hans HAGEN: *ConTeXt, an Excursion*. Version 990527. 27 May 1999. PRAGMA. www.pragma-ade.com/general/manuals/ms-cb-en.pdf.
- [13] Jean-Michel HUFFLEN: “Extracting Information from L^AT_EX Source Files”. *TUGboat*, vol. 43, no. 2, pp. 136–141. 10.47397/tb/43-2/tb134hufflen-extract. 2022.
- [14] Jean-Michel HUFFLEN: “Making an Issue of *Cahiers GUTenberg*”. In: *A Model Kit. Modelling and Implementing Text Typesetting in TeX and Other Systems*. Proc. BachoTeX 2023. April 2023.
- [15] Jean-Michel HUFFLEN: “Making BachoTeX Proceedings”. In: *Composed Thoughts*. Proc. BachoTeX 2023. May 2023.
- [16] Jérémie JUST: “Year 2020 at GUTenberg”. *TUGboat*, vol. 42, no. 1, pp. 12–13. 10.47397/tb/42-1/tb130just-gut. 2021.
- [17] Philip KIME and François CHARETTE: *biber. A Backend Bibliography Processor for biblatex*. Version biber 2.20 (biblatex 3.20). 21 March 2024. ctan.org/pkg/biber.
- [18] Alexis KING: *An Implementation of R7RS in Racket*. 2015. github.com/lexi-lambda/racket-r7rs.
- [19] Yukihiro MATSUMOTO: *Ruby in a Nutshell*. O'Reilly. English translation by David L. Reynolds, Jr. November 2001.
- [20] Andreas MATTHIAS: *The pdfpages Package*. 29 May 2024. ctan.org/pkg/pdfpages.
- [21] Frank MITTELBACH and Michel GOOSSENS, with Johannes BRAAMS, David CARLISLE, Chris A. ROWLEY, Christine DETIG and Joachim SCHROD: *The L^AT_EX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.
- [22] Frank MITTELBACH et Michel GOOSSENS, avec Joannes BRAAMS, David CARLISLE, Chris A. ROWLEY, Christine DETIG et Joachim SCHROD : *L^AT_EX Companion, 2^e édition*. Pearson Education France. Traduction française de [21] par Jacques ANDRÉ, Benoît BELLET, Jean-Côme CHARPENTIER, Jean-Michel HUFFLEN et Yves SOULET. Octobre 2005.
- [23] H.-Martin MÜNCH and Jeffrey P. GOLDBERG: *The lastpage Package*. 3 July 2024. ctan.org/pkg/lastpage.
- [24] Mariusz OLKO and Marcin WOLIŃSKI: *Pakiet polski, wersja 1.3.6*. 25 August 2021. ctan.org/pkg/polksi.
- [25] Oren PATASHNIK: *BIBTEXing*. February 1988. Part of the BIBTeX distribution. ctan.org/pkg/bibtex.
- [26] Éric PICHERAL : « Compte-rendu de lecture ». *La Lettre GUTenberg*, Vol. 31, p. 13–14. Janvier 2006. www.gutenberg-asso.fr/Lettre-GUTenberg-31.
- [27] Sebastian RAHTZ, Heiko OBERDIEK and THE L^AT_EX PROJECT: *Hypertext Marks in L^AT_EX: a Manual for hyperref*. 10 July 2024. mirror.ctan.org/macros/latex/contrib/hyperref/doc/hyperref-doc.pdf.
- [28] Will ROBERTSON, Khaled HOSNY and Karl BERRY: *The X_ET_EX Reference Guide*. 1 March 2024. ctan.org/pkg/xetexref.
- [29] Alex SHINN, John COWAN, and Arthur A. GLECKLER, with Steven GANZ, Aaron W. HSU, Bradley LUCIER, Emmanuel MEDERNACH, Alexey RADUL, Jeffrey T. READ, David RUSH, Benjamin L. RUSSEL, Olin SHIVERS, Alaric SNELL-PYM and Gerald Jay SUSSMAN: *Revised⁷ Report on the Algorithmic Language Scheme*. 6 July 2013. trac.sacrideo.us/wg/raw-attachment/wiki/WikiStart/r7rs.pdf.
- [30] George SPRINGER and Daniel P. FRIEDMAN: *Scheme and the Art of Programming*. The MIT Press, McGraw-Hill Book Company. 1989.
- [31] Richard M. STALLMAN, Roland MCGRATH and Paul D. SMITH: *GNU Make. A Program for Directing Recompilations*. Version 4.4.1. February 2023. www.gnu.org/software/make/manual/make.pdf.

◊ Jean-Michel HUFFLEN
 FEMTO-ST (UMR CNRS 6174)
 & University of Bourgogne
 Franche-Comté
 16, route de Gray
 25030 BESANÇON CEDEX
 FRANCE
 jmhuffle (at) femto-st (dot) fr
<https://members.femto-st.fr/Hufflen-Jean-Michel/en>

Full spectrum litigator: A \TeX -themed workflow for a small litigation law firm

Andrew G. Watters

1 Introduction

I am a lawyer in California, USA, with my own small law firm. I've been in practice for just under 19 years, with my own law firm for 8 years. I am also a self-taught web application programmer with 24 years of experience in the LAMP stack. In that time, I've learned a lot about the challenges facing any small law firm in the area of document generation and management. These challenges include, but are not limited to:

1. Disorganized, ad-hoc workflows that are not at all planned in advance.
2. Proprietary applications that are Windows-only, or even Mac-only.
3. Formatting issues between platforms (Mac and Windows for employees, RHEL for me).
4. Use of ancient templates, usually Microsoft Word, as well as Word macros.
5. Vendor lock-in and subscriptions.
6. Coordinating multiple people on tasks and cases.
7. Integration of various information systems, many of them Windows-only.
8. Email and calendaring when people are used to the convenience and ease of Gmail or the equivalent.

The solution I've come up with (still a work in progress) is one integrated system for handling tasks, time, files, billing, email, calendaring, and reporting. It came about because I was using a Mac and my paralegal was using Windows, and we both needed to access our Samba file server. Because accessing a Samba file server from macOS corrupts files (at least as of 2017), I had to come up with a cross-platform solution for file access, and it was convenient to make it a LAMP web application given my experience in that field.

So the current system is a LAMP web application running PHP and PostgreSQL, formerly with the FPDF library in PHP to generate PDFs (we now use L^AT_EX, as I will show). Email and calendaring are handled by Postfix and Radical, respectively, with access to end users via Thunderbird. The next-generation system we're working on has everything on one screen, and \TeX has become indispensable for this new application.

The current system suffers from a number of shortcomings, starting with the fact that FPDF is not very configurable and certainly cannot be edited

in the same fashion as a normal \TeX file, i.e., from the command line. FPDF is also trial and error with creating cells in documents and outputting them to the browser. Iterating over and over with trial and error is inconsistent with the firm's needs and is also inefficient when the structure of the documents needs to be exact and specified in advance. Enter \TeX .

The current operational use of \TeX in our portal features (1) task orders created from \TeX templates using a custom PHP script that reads the \TeX templates and inserts data from our database, (2) pre-bills and bills using the same method, and (3) Outlook-style email formatting and printing.

2 The \TeX templates

The templates feature a custom, null command, `\php{}`, which is intended to hold variables without being processed. The following (abridged) \TeX template example is from the task order module in our next-generation portal:

```
\documentclass[12pt,letterpaper]{letter}
\usepackage[utf8]{inputenc}
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amssymb}
\usepackage{graphicx}
\usepackage{hyperref}
\usepackage{lastpage}
\usepackage{underscore}
\usepackage{mlmodern}
\usepackage{colortbl}
\usepackage[svgnames]{xcolor}
\usepackage{soul}
\usepackage{setspace}
\hypersetup{
    colorlinks=true,linkcolor=blue,
    filecolor=magenta,urlcolor=blue,
    pdftitle={Task Order},
    pdfpagemode=,
}
\usepackage[margin=1.0in]{geometry}
\usepackage{fancyhdr}
\fancypagestyle{specialfooter}{%
    \fancyhf{}
    \renewcommand\headrulewidth{0pt}
    \fancyfoot[L]{\hrule \vspace{2.0mm} \vfill
        \href{\php{varurl}}{\php{varurl}} \hfill
        \href{mailto:andrew@watters.law}{Andrew} /
        \href{mailto:lindsey@watters.law}{Lindsey} /
        ...
    }
    \fancypagestyle{normalfooter}{%
        \fancyhf{}
        \renewcommand\headrulewidth{0pt}
        \fancyfoot[C]{\thepage}
    }
}
```

```
% Custom highlight colors
\DeclareRobustCommand{\hlpurple}[1]
  {{\sethlcolor{orange}\hl{#1}}}

...
\pagestyle{normalfooter}
\newcommand\tab[1][8.0mm]{\hspace*{#1}}
\newcommand{\php}[1]{}
%\renewcommand*\familydefault{\ttdefault}
% Only if the base font of the document
% is to be typewriter style
\author{Andrew G. Watters}
\title{Letterhead}
\begin{document}
\thispagestyle{specialfooter}
\textbf{\php{varname}} \hfill 555 Twin ...
\hrule
\begin{spacing}{1.5}
\begin{flushleft}
\colorbox{\php{varprecedencecolor}}
{\makebox(\textwidth,40)
\textcolor{white}{\php{vartaskprecedence}}}}\\
TASK ORDER\\
Assigned: \php{vartaskassigneddate}\\
Due: \php{vartaskduedate}\\
Headline: \php{vartaskheadline}\\
Assignment: \php{vartaskassignment}\\
Body: \php{vartaskbody}\\
Tags: \colorbox{teal}...
Meta: \php{vartaskmeta}\\
~\\
Printed on: \today
\end{flushleft}
\end{spacing}
\end{document}
```

As you can see, I've simply started every variable with "var" followed by what I would call the variable in a normal PHP script.

3 The PHP script

Here is the main PHP script (abridged); some discussion follows.

```
<?php
// read .tex template from file system
$file = file_get_contents("task.tex")
or die("Could not read file.");

// put all \php-tagged variables into an array
// with offset position indicated
preg_match("/\\\\\\php{.+?}/",$file,$matches,
PREG_OFFSET_CAPTURE);
```

```
// replace variables with data from input,
// lookup table, or database
$taskid = stripslashes(
strip_tags($_GET['taskid']));

// query database for task info
$connection = new PDO
('pgsql:username=web dbname=watters password=web');
$query = "select distinct tasks.idx as taskid...
$result = $connection->query($query);
foreach ($result as $row) {
    $supervisor = $row['supervisor'];
    $handler = $row['handler'];
    $casename = $row['name'];
    ...
    $caseid = $row['caseid'];
}

// precedence color
switch($precedence) {
    case "0":
        $color = "violet";
        $precedence = "CRITIC";
        break;
    case "1":
        $color = "red";
        $precedence = "URGENT";
        ...
}
// get staff list
$staff = "";
$staff_query = "select idx as staffid ...";
$staff_result = $connection->query($staff_query);
foreach($staff_result as $user_row) {
    $staff .= $user_row['staffid'] . ":" .
    $user_row['handle'] . ":" .
    $user_row['email'] . ","; }
$tagged_staff = explode(",",$staff);
$vartags = ""; $tagemails = "";
foreach ($tagged_staff as $key => $value) {
    $tags = explode(":",$value);
    if (preg_match("/\\b" . $tags[0] . "\\b/", 
        $row['tagged_staff']))
        and (strlen($tags[0] > 0))) {
            $vartags .= $tags[1] . " ";
            $tagemails .= $tags[2] . ","; } }

// lookup table for variables
$name = stripslashes(strip_tags($_GET['name']));
if ($name == "") {
    $name = "Andrew G. Watters, Esq.";
}
$url = stripslashes(strip_tags($_GET['url']));
if ($url == "") {
    $url = "https://www.watters.law"; }
```

```

// arrays of patterns and replacements
$prefix = "/\\\php{"; $suffix = "}/";
$search = array($prefix . "varname" . $suffix,
                $prefix . "varurl" . $suffix, ...);
$replace = array($name, $url, ...);
$newfile = preg_replace($search,$replace,$file);

// create new .tex file
file_put_contents("./tasks/taskorder_" . $taskid
    . ".tex", $newfile)
or die("Could not write taskorder_$taskid.tex.");

// process bar.tex file
$bin = "/home/texlive/2024/bin/x86_64-linux/"
    . "pdflatex";
$opt = "-interaction nonstopmode";
$file = "taskorder_$taskid.tex";
exec("cd tasks; $bin $opt $file")
or die("Could not render PDF.");

// if asked to mail to user, send email
$email = stripslashes
    (strip_tags($_GET['email']));
if ($email == "true") {
    $uid = md5(uniqid(time()));
    $attachment = base64_encode
        (file_get_contents
            ("./tasks/taskorder_$taskid.pdf"))
    or die("Could not load PDF attachment.");
    // get tagged users' emails
    $emails = explode(",",$tagemails);
    foreach($emails as $key => $value) {
        if ($value != "") {
            echo "mailto: $value\n";
            mail($value,
"Task Order - Andrew G. Watters, Esq.",
"--$uid\r\nContent-type:text/plain...");
        }
    }
} else {
    // display PDF in browser
    $filename = "./tasks/taskorder_$taskid.pdf";
    header('Content-type:application/pdf');
    header('Content-disposition: inline; filename="',
        . $filename . "'");
    header('Content-Transfer-Encoding:binary');
    header('Accept-Ranges:bytes');
    @readfile($filename) or die("File not found.");
}
?>

```

In my PHP code, I read the TeX template file into a string. I then specify arrays of patterns and replacements using the built-in `preg_match` function in PHP. As long as the arrays of patterns and replacements are the same length, the patterns will be replaced in the order they appear in the arrays. In other words, the first element of the pattern array is replaced with the first element of the replacement array, and so on.

Once the replacements are made, I write a new TeX file with the correct data. I then render the TeX file with the `pdflatex` command and display the results in the browser, or else email the PDF to the user, depending on user preference.

4 Task orders

Tasks in the law firm are assigned on the home page of the portal. The problem is that the portal is a static web page and there is no easy way to print out particular tasks with all the correct fields and have it be user-friendly. So I implemented the task order feature that emails users a PDF with all the key information that they can print out, if desired:

Andrew G. Watters, Esq.
Litigation and Trial Law Firm
+1 (415) 261-8527

555 Twin Dolphin Dr., Ste. 135
Redwood City, CA 94065
andrew@watters.law

PRIORITY

TASK ORDER

Case: McMillin v. Yorkpro
Assigned: 2024-07-06 09:50:57
Due: 2024-07-13 09:49:16
Assignment: AGW-JSS
Headline: Prepare motion to enforce settlement 664.6
Body: Client's former counsel stipulated to judgment and it was never paid. Prepare motion to enforce under 664.6.
Estimate: 8.0 hours
Tags: AGW JSS
Meta:
Printed on: August 10, 2024

<https://www.watters.law> Andrew / Lindsey / Susanna / Jeramy

Of course, I can also print the task orders to my office copier from the command line as part of this system with an additional command. In this way, I can assign and print a task from anywhere with Internet access, which is helpful when I'm working

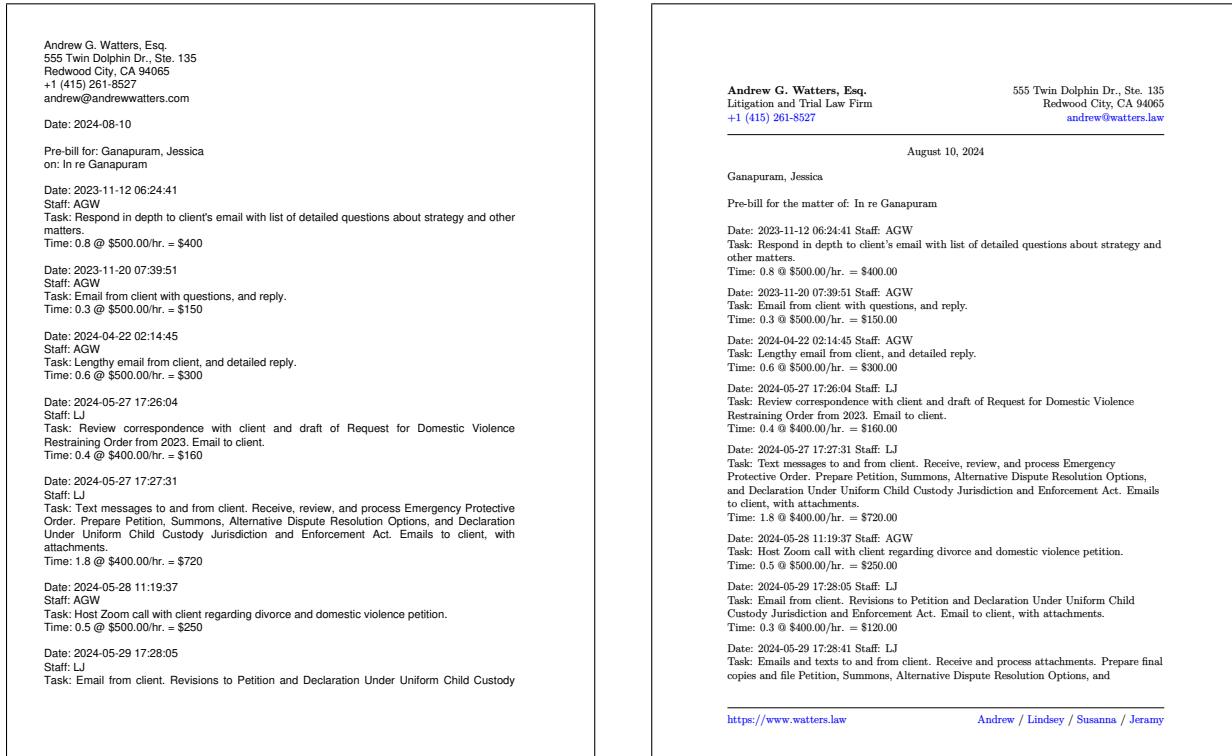


Figure 1: FPDF (left) and TeX (right) output of a pre-bill.

from home or in the event I ever decide to become a digital nomad.

5 Pre-bills and bills

Pre-bills are an opportunity for the clients to see where they stand with their accounts and what they will owe when the bill is issued. This is a commonly used feature and is critically important in a law firm. I'll display the FPDF result and the TeX result side-by-side in figure 1.

Although FPDF is faster than TeX, the TeX result is much more professional-looking and more easily formatted.

6 Outlook-style email printing

On this module, I integrate IMAP commands and data with TeX to retrieve data from my email server and print it to a PDF using the same method as above (figure 2). It still requires some work to get the formatting correct, which is in progress.

7 Future work

Future uses that we're working on include (1) generating actual pleadings and other legal documents from templates using existing data in our database, and (2) printable calendars and reports showing deadlines and similar information. This is an ideal use case for TeX for several reasons, including that it's free

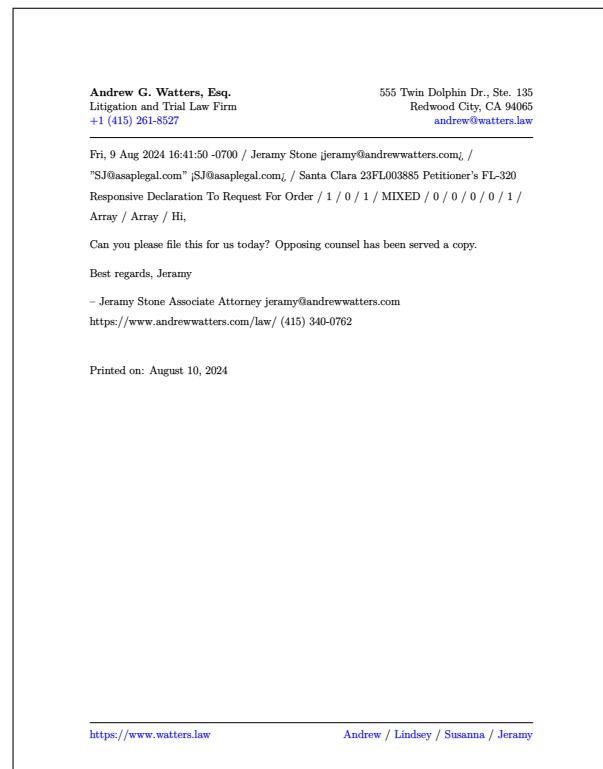
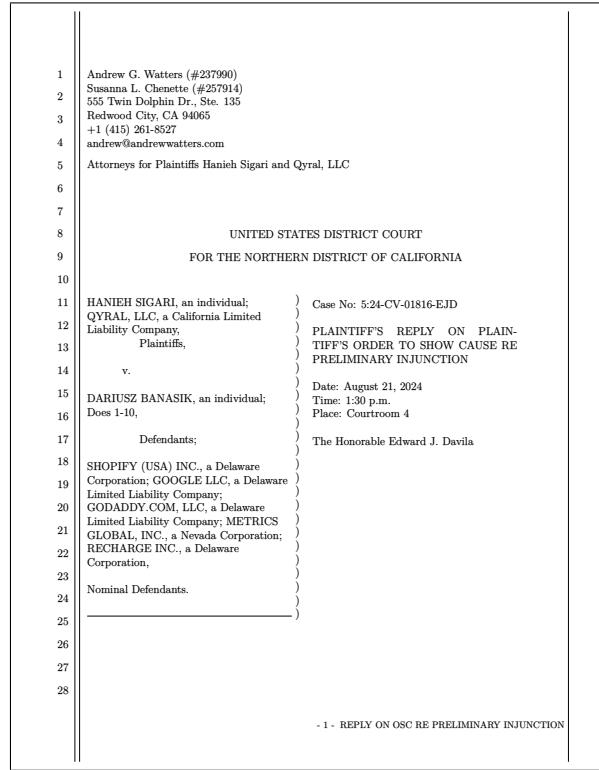


Figure 2: Sample email PDF.

and open source. A sample of what may be possible follow:



8 Conclusion

TeX is the greatest free software of all time for us, and it integrates nicely with the LAMP web application stack that we have been using for nearly ten years. I find it straightforward to integrate into my web application because it's editable in Vim and can process relatively quickly.

Challenges include that rendering speed is not optimal for a web-based user interface. However, as pointed out during the Q&A for my presentation at TUG 2024, it may be possible to pre-render the most time-consuming sections of the template in advance in order to accelerate the responsiveness of the application. I intend to explore that in the near future.

- ◊ Andrew G. Watters
555 Twin Dolphin Dr., Ste. 135
Litigation and Trial Law Firm
Redwood City, CA 94065
+1 (415) 261-8527
andrew (at) andrewatters dot com
<https://andrewwatters.com>

Expanding hyphenation patterns across Slavic languages

Ondřej Sojka

Abstract

So far, TeX hyphenation patterns, even for related languages, have been developed separately for each language, splitting scarce human resources. As languages develop and (especially) English terms creep into formerly monolingual texts, hyphenation patterns, especially for medium- and low-resource languages which often lack quality generated patterns, are due for an update. In this article, we explore the possibilities for transfer learning of hyphenation rules between related Slavic languages.

We present new hyphenation patterns for multiple Slavic languages, developed using transfer learning from various sources.

1 Motivation

Hyphenation patterns play a crucial role in typesetting and text layout, particularly for languages with long words or narrow text columns. They ensure proper word breaks at line ends, improving readability and aesthetics of printed or digital text. Good hyphenation patterns contribute to more uniform text distribution, reducing the occurrence of large gaps between words or excessive hyphenation, making reading more pleasant.

The quality of available hyphenation patterns across Slavic languages varies, with low- and medium-resource languages being impacted the most. Often, the only patterns available are ones made by hand, without the pattern generation program Patgen [2], more than a decade ago. These are insufficient, especially considering the mediocre generalization capabilities of Patgen.

Hyphenation patterns in Slavic languages are, however, *syllabic* and syllabification is very similar across languages.

Pattern generation is also a niche topic and the associated know-how is fairly sparsely distributed.

But since syllabification rules and patterns do not vary across languages from the same family, why do we have to develop patterns for each language separately? After all, native speakers of one Slavic language, upon *hearing a spoken word* from a different Slavic language and being provided with the written form, can hyphenate it correctly.

If we can acquire text data that express the spoken form of the word, we should be able to generate patterns that hyphenate as well as such a native speaker.

Algorithm 1 Transfer hyphenations between word forms

Require: hyphenated (hyphenated word in source script), target (unhyphenated word in target script)

Ensure: best_result (hyphenated word in target script)

```

1: function TRANSFERHYPHENs(hyphenated, target)
2:   num_hyphens ← COUNTHYPHENs(hyphenated)
3:   possible_positions ← {1, ..., len(target) - 1}
4:   best_result ← ""
5:   min_distance ← ∞
6:   for hyphen_positions in COMBINATIONS(possible_positions, num_hyphens) do
7:     if FIRST(hyphen_positions) ≠ 0 and LAST(hyphen_positions) ≠ len(target) - 1 then
8:       candidate ← INSERTHYPHENs(target, hyphen_positions)
9:       current_distance ← LEVENSSTEINDISTANCE(hyphenated, candidate)
10:      if current_distance < min_distance then
11:        best_result ← candidate
12:        min_distance ← current_distance
13:      end if
14:    end if
15:   end for
16:   return best_result
17: end function
```

2 International Phonetic Alphabet

The International Phonetic Alphabet (IPA) [1] is a standardized system for representing the sounds of human speech. Created by the International Phonetic Association, it uses Latin-based symbols to uniquely represent phonemes, stress, and intonation across all languages. In our project, IPA serves as a crucial intermediary, providing a **common phonetic representation** that *bridges orthographic differences* between Slavic languages. This allows us to capture phonological similarities that might be obscured by orthographic differences and varied scripts (Latin vs. Cyrillic), enabling effective cross-lingual transfer of hyphenation patterns.

3 Joint IPA-form data preparation

3.1 Data acquisition

To start, we need a dataset of words used in each of the *target languages*¹ with frequency data. Given the importance of replicability and licensing restrictions often placed on proprietary datasets, we settled with a cleaned wordlist of all words from the Wikipedia of each language. We strip the XML tags and clean words that occur relatively more frequently on Wikipedia as part of common article layouts, such

as Table, References, External links and similar, acquiring a *replicable*, relatively clean, wordlist.

3.2 Hyphenation of original word forms

We apply the best available hyphenation patterns for each target language to hyphenate all the words in our frequency word list with a frequency higher than 50 and generate the file *<lang>.wlh*, containing the word list **hyphenated**.

3.3 Transfer of hyphenations to IPA word form

We use **espeak-ng** [4] to convert from the written word form (in either Latin or Cyrillic script) to the form in IPA [1].

The next step is to acquire hyphenated words in IPA by *transferring* the hyphenations from the written (Latin or Cyrillic) form to IPA. We use Algorithm 1 to transfer the hyphenations.

This approach is computationally expensive, but is highly parallelizable and therefore not a problem on modern hardware.

4 Joint IPA-form pattern generation

To generate patterns that hyphenate across languages in IPA, we first need to decide what data to use. If we were to weigh data from each language in the training set equally, considering that any machine learning model generally can be only as good as its training data, we would get mediocre patterns.

¹ Target languages are all Slavic languages for which some hyphenation patterns currently exist and which have their own mutation of Wikipedia. Only languages which pass evaluation will be proposed for inclusion in **hyph-utf8** [3].

Patterns are indeed able to learn the IPA dataset, as shown by the results of a run with correct optimized parameters: good: 99.84%, bad: 0.13%, missed: 0.16%.

4.1 Ground truth data for evaluation of data mixes

To decide on the mix, we need data to evaluate the quality of a given language-specific pattern set. To do this, we acquire ground truth data from various sources—in order of preference: language institutes, dictionaries, wiktionaries, human labelers, etc. It is disappointingly rare to find hyphenations in orthographic dictionaries.

4.2 Mixing training data for joint IPA-form pattern generation

To generate high-quality patterns that effectively hyphenate across Slavic languages in IPA form, we employ a strategic approach to mixing training data. Our process involves the following steps:

1. **Initial sampling:** We randomly sample five sets of weights from the possible weight set. Each weight corresponds to the importance given to a specific language's training data.
2. **Model fitting:** Using these initial weight sets, we fit a random forest model. This model learns the relationship between the weight combinations and the quality of the resulting patterns.
3. **Guided sampling:** The random forest model is then used to guide further sampling of weight combinations. This approach allows us to explore the weight space more efficiently, focusing on areas that are likely to yield better results.
4. **Evaluation:** For each set of weights, we generate patterns and evaluate them using a custom scoring function. The score is calculated as good – bad × 5, where ‘good’ represents correctly placed hyphenation points and ‘bad’ represents incorrectly placed ones. This scoring method heavily penalizes incorrect hyphenations while rewarding correct ones.
5. **Selection:** After exploring a predetermined number of weight combinations, we select the set that produces the highest score.

This method allows us to efficiently search the space of possible weight combinations and find an optimal mix of training data from different Slavic languages.

5 Final language-specific pattern generation

As the final step, we convert each of the target language frequency datasets to IPA, hyphenate them with the joint patterns and use algorithm 1 on the previous page to transfer the hyphens to the target language. Having a well-hyphenated wordlist, we run Patgen with a custom parameter set inspired by previously published correct-optimized patterns [5], and generate the final language-specific patterns.

6 Evaluation

To evaluate the quality of the resulting patterns, we turn from machines back to humans. Native speakers of every target language will be presented with sets of 100 randomly shuffled hyphenations and will be asked to decide which hyphenation they find better. For languages in which the improvement has cleared the threshold of statistical significance, we will propose their inclusion into `tex-hyphen` [3], the de facto canonical repository of hyphenation patterns.

References

- [1] International Phonetic Association. *Handbook of the International Phonetic Association: A Guide to the Use of the International Phonetic Alphabet*. Cambridge University Press, Cambridge, 1999.
- [2] 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
- [3] A. Reutenauer, M. Miklavec. T_EX hyphenation patterns. hyphenation.org/
- [4] J. Reynolds. eSpeak NG, 2016. github.com/espeak-ng/espeak-ng
- [5] P. Sojka, O. Sojka. New Czechoslovak hyphenation patterns, word lists, and workflow. *TUGboat* 42(2):152–158, 2021. doi.org/10.47397/tb/42-2/tb131sojka-czech

◊ Ondřej Sojka
Faculty of Informatics, Masaryk Univ.,
Brno, Czech Republic
454904 (at) mail dot muni dot cz
ORCID 0000-0003-2048-9977

Packaging Arsenal fonts for (Xe /Lua) L^AT_EX

Boris Veytsman

Typography is an art, and as any art, reflects peoples' lives, tastes, convictions, and views. An important part of the reality today is the process of decolonization, when groups previously not heard acquire voices. This process inspires an interest in the artistic traditions of formerly underrepresented groups, including their writing traditions. The interplay of politics, social movement, and typography in the decolonization era was among the most discussed topics at the recent *Face/Interface* conference at Stanford (a report on this conference can be found in Veytsman (2024a)).

It is not surprising that Ukraine, which was one of the first conquests of Russia, and now struggles to defend its independence, tries to rethink its traditions as different from those of the oppressors. A good illustration of this process may be the recent change of signage for Kyiv metro stations. In Figure 1 is shown the old station name (*Leo Tolstoy Place*) and the new name (*The Place of Ukrainian Heroes*). The change does not just involve the transition from the Russian realities (Leo Tolstoy, while being a great writer, has little to do with Ukraine), but also the transition to a quite distinctive Ukrainian typography, proudly showing off its "Ukraineness".

The decolonization of art is important not just for the decolonized people, but for the rest of us as well, making our common inheritance richer. I wanted to participate in this process by including in TeX distributions a distinctively Ukrainian font. While



The old signage (Wikimedia Commons, 2024)



The new signage and its author Bohdan Hdal (Яковенко, 2024)

Figure 1: Signage for a Kyiv metro station



Figure 2: Andrij Shevchenko, from www.ukrainian-type.com/fontarsenal/

there is a number of such fonts today, most of them are commercial. Free Ukrainian fonts are often released under licenses incompatible with TeXLive. Fortunately there was a high quality font recently released under Open Font License (OFL). It has a quite interesting history, which we will briefly touch here.

One of the influential cultural establishments of Ukraine is Mystetskyi Arsenal, "The Arsenal of Arts" (artarsenal.in.ua/en/), a museum and art exhibition complex in Kyiv. By the way, the word "Arsenal" itself has a multicultural history, having the origin in Arabic *dār-aṣ-ṣinā'*, "the house of art/industry", and probably introduced into modern European languages due to the "Venetian Arsenal", a complex of shipyards and armories dating to early 12th century (I am grateful to Enrico Gregorio who explained this etymology to me). In 2011 Mystetskyi Arsenal and STAIRSFOR studio conducted a competition for a modern, business-like font in Ukrainian traditions (www.ukrainian-type.com/about/) (the idea of the competition belongs to Mykhaylo Il'ko). The first prize was ₴100 000 (about €10 000 at that time). According to the competition rules, the winner ought to change the title of the font to *Arsenal* and release it under the OFL.

The competition was won by Andrij Shevchenko (Figure 2). His font has narrow proportions, moderate aperture, and observable contrast. It is clear, neutral and easy to read.

Since the font was released under the OFL, the magic of free software did its work: several people took efforts to extend it. Additional letters were added by Alexei Vanyashin, Marc Foley & cyreal.org, and Vietnamese support was provided by Nhun Nguyen. Now it is quite a mature font, with 715 glyphs (github.com/alexeiava/Arsenal).

Theorem 1 (Residue Theorem). Let f be analytic in the region G except for the isolated singularities a_1, a_2, \dots, a_m . If γ is a closed rectifiable curve in G which does not pass through any of the points a_k and if $\gamma \approx 0$ in G then

$$\frac{1}{2\pi i} \int_{\gamma} f = \sum_{k=1}^m n(\gamma; a_k) \text{Res}(f; a_k).$$

Theorem 2 (Maximum Modulus). Let G be a bounded open set in \mathbb{C} and suppose that f is a continuous function on G^- which is analytic in G . Then

$$\max\{|f(z)| : z \in G^-\} = \max\{|f(z)| : z \in \partial G\}.$$

(a) `math=arsenal+kpsans` option

Theorem 1 (Residue Theorem). Let f be analytic in the region G except for the isolated singularities a_1, a_2, \dots, a_m . If γ is a closed rectifiable curve in G which does not pass through any of the points a_k and if $\gamma \approx 0$ in G then

$$\frac{1}{2\pi i} \int_{\gamma} f = \sum_{k=1}^m n(\gamma; a_k) \text{Res}(f; a_k).$$

Theorem 2 (Maximum Modulus). Let G be a bounded open set in \mathbb{C} and suppose that f is a continuous function on G^- which is analytic in G . Then

$$\max\{|f(z)| : z \in G^-\} = \max\{|f(z)| : z \in \partial G\}.$$

(b) `math=kpsans` option

Theorem 1 (Residue Theorem). Let f be analytic in the region G except for the isolated singularities a_1, a_2, \dots, a_m . If γ is a closed rectifiable curve in G which does not pass through any of the points a_k and if $\gamma \approx 0$ in G then

$$\frac{1}{2\pi i} \int_{\gamma} f = \sum_{k=1}^m n(\gamma; a_k) \text{Res}(f; a_k).$$

Theorem 2 (Maximum Modulus). Let G be a bounded open set in \mathbb{C} and suppose that f is a continuous function on G^- which is analytic in G . Then

$$\max\{|f(z)| : z \in G^-\} = \max\{|f(z)| : z \in \partial G\}.$$

(c) `math=iwona` option

Figure 3: Mathematical typesetting. The sample text is taken from (Mittelbach and Fischer, 2023).

\LaTeX support (Veytsman, 2024b) was almost embarrassingly easy to write due to the great `fontspec` package (Robertson and The \LaTeX Project Team, 2024). *Arsenal* \LaTeX has usual the options `default` and `sfdefault` to specify the roman or sans serif font as the default, respectively. The option `Scale` allows to select the scaling of the font. The package provides the usual commands to select the normal font, *italic*, **bold**, and **bold italic** versions. The font does not have small caps, but its swash version has all four variants: **NORMAL**, **ITALIC**, **BOLD**, and **BOLD ITALIC**. These variants are selected by the declaration `\swfamily` or the command `\textsw`. The font has relatively rare currency symbols, `\texthryvnia ₴`, `\texttugrik ₴`, `\texttenge ₮`, `\texttruble ₽`, as well as printer's devices `\textaldine ☈`, `\textsmilewhite ☉`, `\textsmileblack ☉`. The package provides commands for them.

Mathematical support currently is rather experimental (see Figure 3) The package provides four variants of math support. The option `math=arsenal+kpsans` uses letters from Arse-

nal fonts, adding missing symbols from KpMath Sans font (Flipo, 2024) (Figure 3a). This is the default when *Lua \LaTeX* is used. Unfortunately, due to a bug in *Xe \LaTeX* , the spacing seems to be off under *Xe \LaTeX* . The option `math=kpsans` uses KpMath Sans for all math (Figure 3b). The option `math=iwona` uses the condensed light Iwona mathematical font (Nowacki, 2010) with the interface (Veytsman, 2024c) (Figure 3c). This is the default for *Xe \LaTeX* . The fourth option, `math=none` is intended for those who want to experiment with their own approaches to math support.

The package is written in the `expl3` language to take advantage of the $\text{\LaTeX}3$ option handling.

Of course, this article is typeset with Arsenal. To give some impression of how Ukrainian looks when typeset with this font, we show a poem by the modern Ukrainian poet Serhiy Zhadan (Figure 4).

Recently Arsenal was chosen as the font of the month for the November 2023 issue of *La Lettre GUTenberg* (Chupin, 2023) (Figure 5). It is both an honor and an opportunity to see the font used in “real life”. Arsenal seems to pass this test rather well. This



Голос із того боку ріки.
Ранньої осені тихий щоденник.
Сидять, поклавши на стіл кулаки,
сільський учитель і сільський священник.

Тепло багатодітних родин.
Ловиться присмак чогось гіркого.
— Сьогодні нікого, — каже один.
— І в мене, — інший говорить, — нікого.

Гілка знадвору хилиться враз.
Тисне голка чужої кривди.
— Піду, — каже перший, — прогрію клас.
Почекаю, може хтось прийде.

— Давай, — говорити інший, — іди.
Доші над річищем, наче квіти.
Попереду вечір і холоди.
Потрібно чекати.
Потрібно гріти.

Сергій Жадан, 07.08.21

Figure 4: A poem by Serhiy Zhadan (Жадан, 2023)



Figure 5: La Lettre GUTenberg, 51, novembre 2023

again proves that national traditions in art have meaning outside the nation's borders, being the legacy of us all.

Bibliography

- Maxime Chupin. *La fonte de ce numéro : Arsenal*. *La Lettre GUTenberg*, 51:59–62, 2023. [publications.gutenberg-asso.fr/lettre/article/view/142/151](https://gutenberg-asso.fr/lettre/article/view/142/151).
- Daniel Flipo. *The kpfonts-otf package*. OTF version of the Kp-fonts, 2024. ctan.org/pkg/kpfonts-otf.
- Frank Mittelbach and Ulrike Fischer. *The LATEX Companion: Parts I & II*. Addison-Wesley, third edition, 2023.
- Janusz Marian Nowacki. *The iwona package*. A two-element sans-serif font, 2010. ctan.org/pkg/iwona.
- Will Robertson and The LATEX Project Team. *The fontspec package*. Advanced font selection in XeLaTeX and LuaLaTeX, 2024. ctan.org/pkg/fontspec.
- Boris Veytsman. Face/Interface 2023 conference: Global type design and human-computer interaction. *TUGboat*, 45(1):7–10, 2024a. ISSN 0896-3207. doi.org/10.47397/tb/45-1/tb139veytsman-face.
- Boris Veytsman. *The arsenal package*. Open Type font by Andrij Shevchenko, 2024b. ctan.org/pkg/arsenal.
- Boris Veytsman. *The iwonamath package*. LATEX support for scaled Iwona math fonts, 2024c. ctan.org/pkg/iwonamath.
- Wikimedia Commons. File:Площа Льва Толстого назва.jpg – wikimedia commons, the free media repository, 2024. commons.wikimedia.org/w/index.php?title=File:%D0%9F%D0%BB%D0%BE%D1%89%D0%B0_%D0%9B%D1%8C%D0%B2%D0%BO_%D0%A2%D0%BE%D0%BB%D1%81%D1%82%D0%BE%D0%B3%D0%BE_%D0%BD%D0%BD%D0%B7%D0%B2%D0%BO.jpg&oldid=859561010.
- Сергій Жадан. Скрипниківка. Meridian Czernowitz, 2023.
- Ірина Яковенко. На станції метро «Площа Українських Героїв» встановили літери для нової назви. *The Village Україна*, April 2024. [village.com.ua/village/city/city-news/349665-na-stantsiyi-metro-ploscha-ukrayinskikh-heroyiv-vstanovili-novu-nazvu](https://www.village.com.ua/village/city/city-news/349665-na-stantsiyi-metro-ploscha-ukrayinskikh-geroyiv-vstanovili-novu-nazvu).
- ◊ Boris Veytsman
TeX Users Group
borisv (at) lk dot net
<https://borisv.lk.net>

dynMath: Underlying principles of the design

Abdelouahad Bayar

Abstract

This article presents the basic principles underlying the design and development of **dynMath**, a package that supports dynamic mathematical symbols. The focus is on the interaction between L^AT_EX and PostScript via the T_EX \special primitive, and in particular the direct use of a dynamic PostScript Type 3 font in the L^AT_EX source.

1 Introduction

Electronic documents, especially scientific ones, are typeset using static and/or dynamic characters. The mathematical formula is always the most suitable example for highlighting the subject. Mathematical variable-sized symbols, such as delimiters (parentheses, braces, radicals, etc.), are a good way to make the subject concrete.

When we talk about scientific document processing, we think first and foremost of (L^A)T_EX in its various implementations: T_EX [5], L^AT_EX [6], LuaT_EX [7], etc. Dynamic symbols such as delimiters and others are supported by (L^A)T_EX but in some cases the properties of *optical scaling*, *uniformity* of shape, *right-sizing* and *metal likeness* are not respected. The **dynMath** package [4] has been developed with the aim of supporting such characteristics and thus enhancing and improving the typesetting quality of (L^A)T_EX.

(L^A)T_EX offers the possibility of interacting with PostScript [1] via the T_EX \special primitive. The latter makes it possible to insert and manipulate PostScript code in (L^A)T_EX through the dvips driver [8] while generating PostScript from the dvi files. We have used this mechanism to handle a dynamic Type 3 [1] font in T_EX, thus enabling dynamic mathematical symbols to be supported by the **dynMath** system. The way in which this approach of interaction is used is unusual in the development of (L^A)T_EX packages. For this reason, we believe it will be interesting to present details of the implementation process. We note that the same work was done when the development of **dynMath** was launched in 2016 [2]. The resumption of such work is justified by the change that has taken place in the implementation.

This paper is organized as follows. In Section 2, the overall layout of the **dynMath** system is given. In Section 3, some details of **dynMath** in terms of the Type 3 font are presented. In Section 4, the way in which **dynMath** supports dynamic mathematical symbols in terms of T_EX programming and interac-

tion with the dynamic font Type 3 is studied. The paper ends with some conclusions and perspectives.

2 dynMath: The layout

The **dynMath** system is now in its basic state. It contains the minimum necessary files to operate, namely **dynMath.sty** and **dynMath.tps**.

- **dynMath.sty**: this is the L^AT_EX package itself. It contains the definition of the macros required to support the mathematical variable-sized symbols.
- **dynMath.tps**: this is the specification of a PostScript Type 3 font parameterized to draw mathematical symbols with dimensions and shapes satisfying given contexts.

Some details of the two files will be seen below to give an idea on how they work. A part highlights the interaction between L^AT_EX and PostScript Type 3.

3 dynMath: The font

3.1 PostScript inside L^AT_EX

The requirements for supporting dynamic mathematical symbols are identified in [2]. The PostScript language and PostScript Type 3 fonts are recognized as suitable to provide a solution.

Natively, (L^A)T_EX has an interface to fonts specified in METAFONT. This is achieved through tfm files. These communicate information about the dimensions, in a definitive static way, of the characters which will appear in the document to be printed. METAFONT is a compiled language and does not allow for manipulating the characteristics of characters at printing time.

It is also possible to use a Type 3 font as a virtual font. Even if a Type 3 font fully uses the PostScript language and can be parameterized in a flexible way, it will not be able to offer support for dynamism via virtual fonts because the latter are seen by T_EX as if they were METAFONT fonts.

(L^A)T_EX supports handling the PostScript code as a literal in the \spacial command in different ways, depending on the scope of the code in the generated PostScript document. The most important thing is that the PostScript variables, in these literals, can be evaluated based on T_EX variables whose values are determined at a given time and in a given context. This PostScript code can in particular be a dynamic (parameterized) PostScript Type 3 font.

The PostScript Type 3 font is specified in the file **dynMath.tps**. It is a font which respects the Type 3 specification but it is included in the macro primitive \spacial and having a global PostScript scope:

```
% Content of ``dynMath.tps''
\special{!}
<PostScript Type 3 specification of dynmath font>
}
```

This is an interaction between L^AT_EX and PostScript in which the Type 3 font is inserted and seen throughout the document generated by L^AT_EX via the dvips driver.

3.2 Symbols in table and encoding

Any font (PostScript in particular) defines a table of its character layout: graphics and code. The Type 3 font in `dynMath.tps` is called `dynMath`. We used `cmex10` (see Table 1) to build the layout of `dynMath` symbols. The `dynMath` layout is shown in Table 2. Because `dynMath` is dynamic, the symbol appears only once in the table. However, the symbol is parameterized to meet the required dimensions in a given context.

The symbols coded from 70 to 97 in the `cmex10` layout table come in two graphic versions (one for the mathematical mode `\scriptstyle` and the other for `\displaystyle`). We think that these signs, of T_EX math class one (large operators), will remain in these two size cases (obviously referenced by their relative (L^A)T_EX commands). They will not be supported in `dynMath` except for the integral signs \oint and \int . This is because an integral sign with a height greater than or equal to the mathematical quantity to be integrated looks better than the opposite case.

In [5], four fonts are mainly identified by the values `\textfont0`, `\textfont1`, `\textfont2` and `\textfont3` as the METAFONT fonts `cmr10.mf` (family 0), `cmmi10.mf` (family 1), `cmsy10.mf` (family 2) and `cmex10.mf` (family 3) respectively. We are interested in dynamic (extensible) symbols. In (L^A)T_EX, dynamism is managed by using different fonts depending on the context. To explain the concept, we will use the symbols “(”, “⟨”, “√” and “^”. We consider the contents of the file `plain.tex` as reference.

- `\delcode`(`=028300`: This means that the parenthesis “(” is a delimiter, of which the smallest variant is taken from family 0 at position 28x (40 in decimal) and the wide variant is in family 3 at position 00x.
- `\def\langle{\delimiter"426830A }`: This defines the symbol “⟨” as a delimiter of class 4 (opening delimiter), accessible via the `\langle` macro. The smallest variant is in family 2 at position 68x (104 decimal) and the largest is in family 3 at position 0Ax (10 decimal).
- `\def\sqrt{\radical"270370 }`: This defines the radical symbol “√” as a variable symbol whose smallest variant is in family 2 at position

70x (112 decimal) and the large variant is in family 3 at position 70x, accessed as the `\sqrt` macro.

- `\def\widehat{\mathaccent"0362 }`: This defines the wide hat symbol “^” by means of the command `\widehat`, of class 0 (ordinary) and of which the smallest variant is in the family 3 at position 62x (98 decimal).

For the left parenthesis symbol, the smallest variant is encoded in the font `cmr10.mf` at position 40. The large variant with its various standalone instances is encoded in the font `cmex10.mf` at positions 0, 16, 18 and 32. The compound version is built from characters in the same font `cmex10.mf` at positions 48, 64 and 66 (repetitive character). The font `dynMath` is dynamic and so any symbol, such as the parenthesis, must appear only once in the layout table. The code is that of the first occurrence of the symbol in `cmex10.mf`, i.e., position 0 (see Table 2). To parameterize the parenthesis and thus support dynamism, we consider the encoding of the smallest variant, that in `cmr10.mf` which is relative to family 0.

There are other symbols whose parameterization is based on their appearances in the `cmr10.mf` font such as “)”, “[”, “]”, etc. Concerning the symbols “⟨” and “√” for example, they are in positions 10 and 112 respectively in `dynMath` (see Table 2) and their parametrizations are taken from the font `cmsy10.mf` relative to family 2. As for the sign “^”, its position in `dynMath` is that in `cmex10.mf` and its parameterization base is of the smallest variant and taken from the same font, namely `cmex10.mf`.

Roughly speaking, it's the encoding bases of the small symbol variants that are parameterized to support dynamism. Consider a symbol S . Its appearance in `dynMath` in Table 2, is of the form S_f^c with c designating the layout order number and f representing the family used as a basis for parameterization. Specifically, the opening bracket, the opening angle bracket and the wide hat are shown in Table 2 as $[_0^2$, \langle_2^{10} and $\widehat{}_3^{98}$.

We transformed the fonts `cmr10.mf`, `cmsy10.mf` and `cmex10.mf` using METAPOST to PostScript code at a body size of 1000 units, serving as a basis for parameterization of dynamic mathematical symbol encoding, via the following commands:

```
mpost '&mfplain \mode=localfont; \
mag=100.375; input cmr10.mf'
mpost '&mfplain \mode=localfont; \
mag=100.375; input cmsy10.mf'
mpost '&mfplain \mode=localfont; \
mag=100.375; input cmex10.mf'
```

Table 1: Math extension font layout showing `cmex10`

	0	1	2	3	4	5	6	7
'00x	0 (1)	2 [3]	4 [5]	6 [7]
'01x	8 {	9 }	10 <	11 >	12 	13 	14 /	15 \
'02x	16 (17)	18 (19)	20 [21]	22 [23]
'03x	24 [25]	26 {	27 }	28 <	29 >	30 /	31 \
'04x	32 (33)	34 [35]	36 [37]	38 [39]
'05x	40 {	41 }	42 <	43 >	44 /	45 \	46 /	47 \
'06x	48 (49)	50 [51]	52 [53]	54 	55
'07x	56 (57)	58 (59)	60 {	61 }	62 '	63
'10x	64 (65)	66 	67 	68 <	69 >	70 □	71 □
'11x	72 §	73 ʃ	74 ⊙	75 ○	76 ⊕	77 ⊕	78 ⊗	79 ⊗
'12x	80 Σ	81 Π	82 ʃ	83 ∪	84 ∩	85 ⊕	86 Λ	87 ∨
'13x	88 Σ	89 Π	90 ʃ	91 ∪	92 ∩	93 ⊕	94 Λ	95 ∨
'14x	96 II	97 II	98 ^	99 ^	100 —	101 ~	102 ~	103 ~~
'15x	104 [105]	106 [107]	108 [109]	110 {	111 }
'16x	112 √	113 √	114 √	115 √	116 √	117 	118 Γ	119
'17x	120 ↑	121 ↓	122 ↶	123 ↷	124 ↶	125 ↷	126 ↑↑	127 ↓↓

There's a special case (as there may be more to come) for the opening and closing brace symbols. These are not based on existing fonts for parameterizing, but have been newly designed to meet the metal-likeness concept.

3.3 Parameterizing

Dynamic symbols are parameterized in the font to meet extension requirements. Two categories of characters are identified, depending on whether the dynamic parts are delimited by *straight lines* or *curved lines*. Two types of stretching are identified:

Table 2: dynMath font layout

	0	1	2	3	4	5	6	7
'00x	($\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}$) $\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}$	[$\begin{smallmatrix} 2 \\ 0 \end{smallmatrix}$] $\begin{smallmatrix} 3 \\ 0 \end{smallmatrix}$	$\lceil \begin{smallmatrix} 4 \\ 2 \end{smallmatrix}$	$\rceil \begin{smallmatrix} 5 \\ 2 \end{smallmatrix}$	$\lfloor \begin{smallmatrix} 6 \\ 2 \end{smallmatrix}$	$\rfloor \begin{smallmatrix} 7 \\ 2 \end{smallmatrix}$		
'01x	{ $\begin{smallmatrix} 8 \\ \end{smallmatrix}$ } $\begin{smallmatrix} 9 \\ \end{smallmatrix}$	$\langle \begin{smallmatrix} 10 \\ 2 \end{smallmatrix}$ $\rangle \begin{smallmatrix} 11 \\ 2 \end{smallmatrix}$	$\mid \begin{smallmatrix} 12 \\ 2 \end{smallmatrix}$	$\parallel \begin{smallmatrix} 13 \\ 2 \end{smallmatrix}$	$/ \begin{smallmatrix} 14 \\ 0 \end{smallmatrix}$	$\backslash \begin{smallmatrix} 15 \\ 2 \end{smallmatrix}$		
'02x	16	17	18	19	20	21	22	23
'03x	24	25	26	27	28	29	30	31
'04x	32	33	34	35	36	37	38	39
'05x	40	41	42	43	44	45	46	47
'06x	48	49	50	51	52	53	54	55
'07x	56	57	58	59	60	61	62	$\updownarrow \begin{smallmatrix} 63 \\ 2 \end{smallmatrix}$
'10x	64	65	66	67	68	69	70	71
'11x	$\oint \begin{smallmatrix} 72 \\ 3 \end{smallmatrix}$	73	74	75	76	77	78	79
'12x	80	81	$\int \begin{smallmatrix} 82 \\ 3 \end{smallmatrix}$	83	84	85	86	87
'13x	88	89	90	91	92	93	94	95
'14x	96	97	$\wedge \begin{smallmatrix} 98 \\ 3 \end{smallmatrix}$	99	100	$\sim \begin{smallmatrix} 101 \\ 3 \end{smallmatrix}$	102	103
'15x	104	105	106	107	108	109	110	111
'16x	$\sqrt \begin{smallmatrix} 112 \\ 2 \end{smallmatrix}$	113	114	115	116	117	118	$\Updownarrow \begin{smallmatrix} 119 \\ 2 \end{smallmatrix}$
'17x	$\uparrow \begin{smallmatrix} 120 \\ 2 \end{smallmatrix}$	$\downarrow \begin{smallmatrix} 121 \\ 2 \end{smallmatrix}$	122	123	124	125	$\updownarrow \begin{smallmatrix} 126 \\ 2 \end{smallmatrix}$	$\Downuparrow \begin{smallmatrix} 127 \\ 2 \end{smallmatrix}$

1. Line-based extension: this type of extension is easy and straightforward to support. Examples include the bracket symbol “[”, the up arrow symbol “↑”, etc.
2. Curve-based extension: this extension concerns symbols whose dynamic parts have curved lines. Here, support for dynamism has necessitated the development of a mathematical stretching model (to be published) and an interpolation method that respects obliquity and convexity [3]. Examples include the parenthesis “(”, the brace “{”, etc.

A dynamic symbol is characterized by three parameters: *height* (including *depth*), *width* and *thickness*. The thickness is in some way linked to the characteristics of the writing instrument (pen) or drawing instrument (brush).

The stretching undergone by a dynamic symbol is partly supported by the *dynMath.sty* package and partly by the *dynMath.tps* font. Consider the dynamic symbol S . Let H_S , W_S and E_S be its *height*, *width* and *thickness* respectively. If the symbol is to be stretched by the amount h vertically and w horizontally, then the features in the stretched state will be $H_S + h$, $W_S + w$ and E_S as its *height*, *width* and *thickness* respectively. Thickness is not affected by the extension. It should be noted that the stretching supported by the font is not linear.

We'll call it semi-optical because the thickness remains unchanged. Globally speaking, the thickness also changes, but this is the work of L^AT_EX and the PostScript interpreter.

The concept is clarified in Figure 1. This is the case of the opening parenthesis but we have considered just the upper half, to show how the font takes care of the stretching on its side. Note that the example is computed at 10 font size in T_EX points but scaled linearly 20 times for greater clarity. It is as if the parenthesis at size 200 in T_EX points undergoes stretching of the amounts w horizontally and h vertically.

The thickness was not affected by the stretching. To highlight this, we have considered landmarks with different colors. The upper half of the parenthesis is delimited by two sequences of curves, one on the left and the other on the right. Each sequence is made up of 7 cubic Bézier curves (how we get these sequences is a separate work from the current one). The points shown are the boundary control points of the Bézier curves. Points of the same order in both sequences are of the same color and linked by a segment also of this color. Our mathematical stretching model preserves the same convexity sense, obliquity and thickness. This is expressed by the fact that segments of the same color are of the same length and direction (in the vector sense) in Figure 1a and Figure 1b.

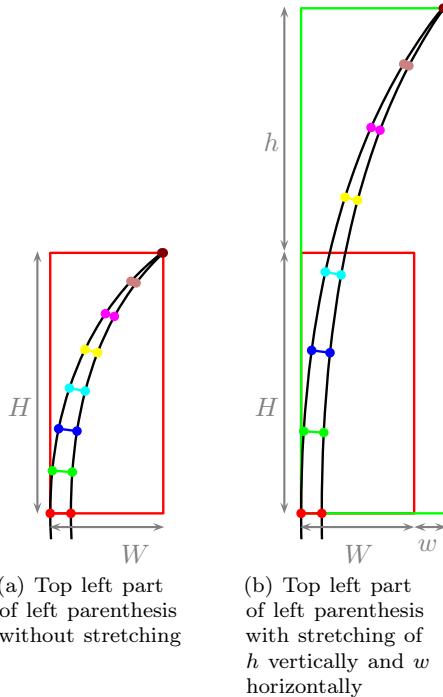


Figure 1: Example of stretching in height h and width w while keeping the same thickness

4 dynMath: the style package

4.1 Useful macros and conventions

The `dynMath.sty` style package defines all the variables useful for internal operations, as well as others used as an interface for interaction with the PostScript Type 3 font `dynMath`. It also defines macros for managing mathematical formulas based on extensible symbols. We have followed a particular way of naming the macros relating to the dynamic symbols in `LATEX`. In `LATEX`, without doubt, the most interesting commands, in term of dynamism, are the primitive `\left` and its counterpart `\right`. The package `dynMath` defines a macro which essentially does the same job as `\left` but operates with the dynamic symbols defined in the PostScript Type 3 font. The general syntax of this macro is:

`\meLeft<delim1> <formula> \meRight<delim2>`

We referred to the normal `LATEX` commands when naming the `dynMath` ones in order to make it easier to use for users accustomed to using `(LA)TEX`. The same names are used, beginning with a capital letter and preceded by “`me`” meaning “metal”. Another example is `\overbrace`, to which corresponds `\meOverBrace` in `dynMath`.

5 Determining extension parameters

The most characteristic stretching parameters are the amount of vertical stretching h , the amount of horizontal stretching w and the size of the font PostScript f_s in which we will typeset the symbol to be stretched. In the case of the `\meLeft` macro, that is, in the case of the delimiters, these three parameters are functions of the mathematical height of a formula, which we will always call h_m . First, we give the idea of calculating h_m . Figure 2 and Figure 3 explain the approach. This concerns the case of two abstract mathematical formulas (just a rectangle with a height, depth and width) one of which is high and the other is deep. A description of the parameters in the figures are as follows:

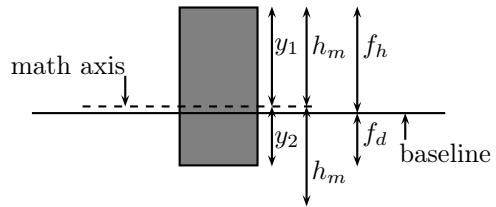


Figure 2: Abstract high mathematical formula

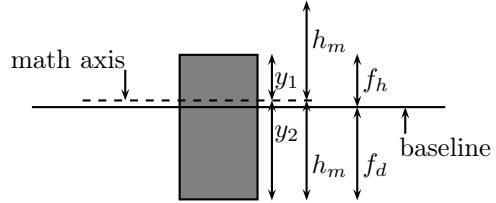


Figure 3: Abstract deep mathematical formula

- f_h : height of formula from the baseline.
- f_d : depth of formula from the baseline.
- y_1 : mathematical height of the formula. It is measured from the mathematical axis to the top of the formula.
- y_2 : mathematical depth of the formula. It is measured from the mathematical axis to the bottom of the formula.
- h_m : mathematical balanced height (depth) of the (balanced) formula. We have that $h_m = \text{Sup}(y_1, y_2)$.

An important point to note is that the handling of stretchable mathematical symbols differs from one category to another. For example, the parameter h_m , which makes sense in the case of delimiters, will not make sense when it comes to the radical (square

Table 3: Characteristics of left parenthesis in PostScript at a 10 unit body size

Parenthesis	Width	Close width	Height	Thickness	Left bearing	Right bearing	Math axis
Normal	3.8688232	2.3218745	5.0000024	0.5833423	0.9942598	0.5526889	2.5000014
big	4.5632816	2.6121009	5.9900241	0.6944529	1.5242052	0.4269755	
Big	5.9521679	3.7974070	8.9900563	0.7638956	1.8027775	0.3519834	
bigg	7.3410694	4.9213804	11.9900885	0.8333385	2.081079	0.33861	
Bigg	7.8966266	5.2128642	14.9901202	0.972224	2.3589455	0.3248169	
Compound	8.7299554	5.5043343	17.9801222	1.1111097	2.9145983	0.3110228	

root) symbol. For the radical, the amount of vertical stretching, for example, depends on the overall height including depth, i.e., $f_h + f_d$.

We only consider the case of the macro `\meLeft`, since the aim is to illuminate the interaction between (L^A)TeX and PostScript Type 3. The value of parameter h_m is half the overall height of the extensible delimiter. To clarify the idea, we take one of the previous figures, Figure 3 for this example, and display on it the left parenthesis useful for delimiting the abstract formula. The result is in Figure 4. The parenthesis delimiter is positioned correctly vertically. However, it has been shifted a little horizontally to the left to give the figure more legibility.

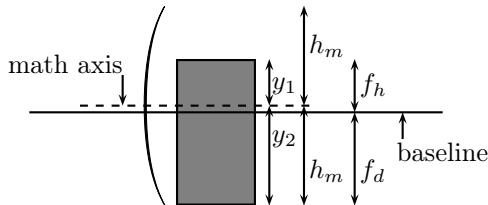


Figure 4: Abstract deep mathematical formula with left parenthesis delimiter

The opening (and closing) parenthesis in TeX comes in five standalone versions, as shown in Figure 5. One or the other is used to delimit a formula, depending on the situation of its mathematical height in comparison with those of the parentheses. When the mathematical height of a formula exceeds that of the standalone parentheses, a three-character compound parenthesis is used. This is made up of the three characters `(`, `\` and `I`, vertically superposed, with the third repeated between the two first as many times as necessary.

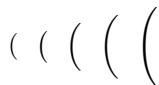


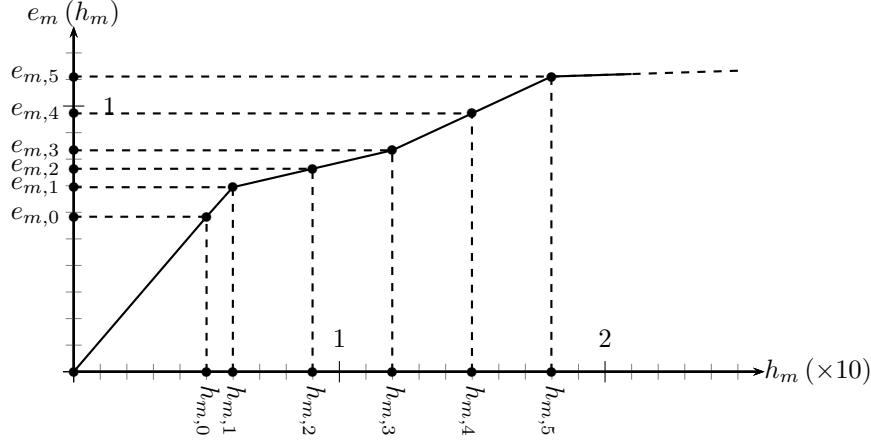
Figure 5: (L^A)TeX standalone left parentheses

Let's adopt a numbered designation for the parentheses, P_0, \dots, P_5 , in the order given in Figure 5. The last, P_5 , is the smallest compound parenthesis, i.e., the compound when the number of occurrences of the repeated character is zero. The parenthesis P_0 represents the smallest variant in standalone parentheses (as we saw before). It is none other than parenthesis number 40 in `cmr10.mf`. It's this parenthesis that we've set in the PostScript Type 3 font `dynMath` to support dynamic parenthesis. Its encoding in PostScript is developed as a function of the two variables w and h (among others) representing horizontal and vertical stretching respectively.

Table 3 shows the most important characteristics of the six parentheses used in (L^A)TeX. One notable parameter of the state of a parenthesis is the thickness e_m . How is this value calculated? In the case of a delimiter to be stretched relative to `\meLeft`, it's the stretching in the vertical direction that attracts attention. This shows that h_m is a key parameter in handling the dynamism of delimiters. For this purpose, thickness is defined as a function of the mathematical height h_m . Let $(h_{m,i})_{i=0}^5$ denote the sequence of mathematical heights of the parentheses P_0, \dots, P_5 . Similarly, $(e_{m,i})_{i=0}^5$ is the sequence of the thicknesses of P_0, \dots, P_5 . The 10pt size is taken as a basis for handling the stretching of the parenthesis symbol. We have the following cases and constraints:

- If $h_m = h_{m,i}$, $i = 0, \dots, 5$ then $e_m = e_{m,i}$.
- If $h_m \in [0, h_{m,0}]$, then e_m is linearly increasing between 0 and $e_{m,0}$.
- If $h_m \in [h_{m,i}, h_{m,i+1}]$, $i = 0, \dots, 4$ then e_m is increasing affinely between $e_{m,i}$ and $e_{m,i+1}$.
- If $h_m \in [h_{m,5}, h_{\max}]$, then e_m is increasing affinely between $e_{m,5}$ and e_{\max} .

The value of the maximum mathematical height taken is $h_{\max} = 1685\text{pt}$. This value represents approximately half the height of an A0 page. As for the thickness corresponding to h_{\max} , determined by experiments based on a certain formulation, it is $e_{\max} = 6.292214230\text{pt}$.

Figure 6: Thickness e_m as a function of h_m

A summary of the cases is shown in Figure 6. Remember that it's the parenthesis P_0 , but with a 1000 unit body size, that is implemented and parameterized in the PostScript Type 3 font. What counts first when typesetting a mathematical symbol, such as the opening parenthesis, is the size f_s of the font. Assuming that for a mathematical height h_m , the thickness is e_m , knowing the thickness e_{1000} of the parenthesis at size 1000 in the PostScript font **dynMath**, then we can determine the size value f_s of the font corresponding to this thickness e_m , i.e., $f_s = (e_m \times 1000) / e_{1000}$.

Let a font size f_s correspond to a thickness e_m which we calculated as a function of h_m . Let us denote by h_{f_s} the height of the parenthesis in the font PostScript Type 3 **dynMath** relative to f_s , without any extension ($w = 0$, and $h = 0$). So we have:

1. If $h_m \leq h_{m,0}$ then $h_{f_s} = h_m$
2. If $h_m > h_{m,0}$ then $h_{f_s} < h_m$.

We assume that h represents the amount of vertical stretching the parenthesis in **dynMath** must undergo to delimit the mathematical formula. For Item 1, the parenthesis obtained has the necessary height to cover the formula. There's no need to stretch this parenthesis, so $h = 0$. On the other hand, in Item 2, we need a vertical extension $h = h_m - h_{f_s}$ for the parenthesis to have the height needed to cover the formula. The horizontal stretching amount w needed will be explained later.

Just like the thickness e_m , other functions are useful and defined according to h_m : the width w_m , the strict or close width (width of the symbol without the left and right bearings) cw_m , the left bearing lb_m and right bearing rb_m .

The function cw_m is important for calculating the amount of horizontal stretching w . For this, we give some detail on its definition. The sequence

$(cw_{m,i})_{i=0}^5$ consists of the close widths of the parentheses P_0 to P_5 . We have:

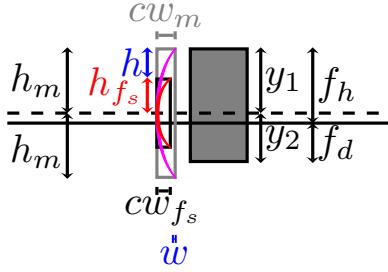
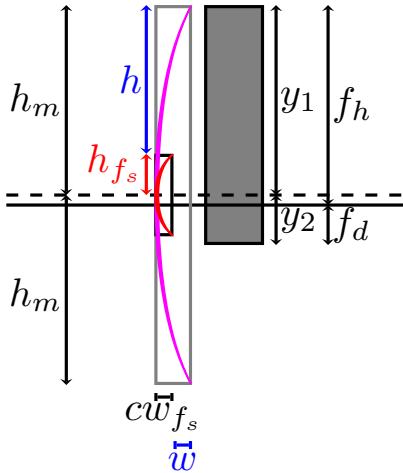
- If $h_m = h_i$, $i = 0, \dots, 5$ then $cw_m = cw_{m,i}$.
- If $h_m \in [0, h_{m,0}]$, then the function is of no interest (see further).
- If $h_m \in [h_{m,i}, h_{m,i+1}]$, $i = 0, \dots, 4$ then cw_m is increasing affinely between $cw_{m,i}$ and $cw_{m,i+1}$.
- If $h_m \in [h_{m,5}, h_{\max}]$, then cw_m is of no interest.

If we reconsider the font size f_s and denote by cw_{f_s} the close width of the parenthesis in the Type 3 font **dynMath** at size f_s , we get the following result: $cw_{f_s} < cw_m$. The horizontal stretching variable w takes on the following values:

1. If $h_m \leq h_{m,0}$ then $w = 0$ (in this case $h = 0$, see Item 1 above).
2. If $h_{m,0} < h_m \leq h_{m,5}$ then $cw_{f_s} < cw_m$ and $w = cw_m - cw_{f_s}$.
3. If $h_m > h_{m,5}$ then $w = \frac{h}{8}$. This is a relationship obtained by experimentation. It differs from one symbol to another. For the brace, for example, it's $w = \frac{h}{16}$.

For further clarification, two illustrations of the last two cases of the above enumeration are in Figures 7 and 8. These figures present information other than that relating to the stretching, vertical h and horizontal w . The meanings of the various parts were given in Figure 2 and Figure 3.

Processing of the left and right bearings is required to correctly position the dynamic symbols around the mathematical formula to be delimited. We need to be aware that the mathematical axis of the symbol written in the **dynMath** font is different from that of the mathematical formula, and so an alignment is necessary. We won't go into the details of these functions here, so as not to overload the article. In a future project, we'll write a book on the detailed implementation of **dynMath**.

Figure 7: Stretching details, $H_4 < h_m < H_5$ Figure 8: Stretching details, $h_m > H_5$

5.1 Dynamism management steps

In this section, the important steps in dynamism management are presented. It should be noted that each macro relating to the extension phenomenon is responsible for managing the relative extension parameters. The need may differ from one macro to another. Consideration of one of them highlights the general concept. The macro used as an example is `\meLeft`. One of the steps in the extension process is interaction with the Type 3 font. We are not going to talk about the `\meLeft` macro in programming terms, but only in an algorithmic sense and in a language as natural and abstract as possible. The definition of this macro is:

```
\def\meLeft#1#2\meRight#3{{ macro definition}}
```

Where:

#1: left delimiter,
#2: formula to be delimited,
#3: right delimiter.

Let's assume that:

`ldel`: represents #1,
`formula`: represents #2,
`rdel`: represents #3.

Before presenting the steps of the `\meLeft` macro, the meanings of some keywords used are given in Table 4.

Table 4: Useful tokens and meaning for dynamism steps presentation

Keyword	Meaning
<code>ldel</code>	left delimiter
<code>rdel</code>	right delimiter
<code>mAxis</code>	mathematical Axis
<code>fbox</code>	formula box
<code>fh</code>	formula height
<code>fd</code>	formula depth
<code>fw</code>	formula width
<code>hm</code>	height mathematical
<code>lth</code>	left thickness
<code>fs</code>	font size
<code>symWidth</code>	symbol Width
<code>fdelb</code>	formula delimiter box

The main steps of `\meLeft` are:

1. Determine the current math style: `style`
2. In `style`:
 - Determine the height of the mathematical axis: `mAxis`.
 - Put `formula` in `fbox`.
3. Determine the dimensions of `fbox`:
 - Height: `fh`
 - Depth: `fd`
 - Width: `fw`
4. Determine the mathematical height `hm`: $hm = \sup(fh - mAxis, fd + mAxis)$
5. Based on `hm`, determine the thickness of the left dynamic symbol `ldel`: `lth`.
6. Based on `lth`, determine the size `fs` of the PostScript font `dynMath` to write the delimiter `ldel`.
7. In terms of `fs` and `hm` determine:
 - The vertical stretching amount `h`.
 - The horizontal stretching amount `w`.
 - The delimiter width `symWidth`.
8. Process the box `fdelb` which will contain the extensible PostScript delimiter:
 - Write in `fdelb` the special: `\special{" leftSpecial }"`.
 - In `{ leftSpecial }`:
 - Align the mathematical axis of the symbol `ldel` according to the font `dynMath` at size `fs` with the mathematical axis `mAxis` of `formula`.
 - Write `ldel` with respect to the font `dynMath` at size `fs` from the coordinates $(0, 0)$.

9. Set the dimensions of `fdelb`:
 - Width at `symWidth`.
 - Height at $(\text{hm} + \text{mAxis})$.
 - Depth at $(\text{hm} - \text{mAxis})$.
10. Adjust the position of `fdelb` by kerning in order to adjust the left bearing of `lde1`.
11. Insert the contents of the `fdelb`.
12. Adjust the right bearing of `lde1` by kerning.
13. Insert `formula`.
14. Repeat steps 5 to 12 for the `rde1` delimiter.

6 Conclusions

We have given an idea on the principles of interaction between (I^A)T_EX and a PostScript Type 3 font. This is the basis for the support of dynamic mathematical symbols. The idea is presented in special cases and not completely detailed. In the near future, we will publish a book detailing the basics and all the implementation cases of `dynMath`.

References

- [1] Adobe Systems Incorporated. *PostScript Language Reference Manual*. Adobe Systems Incorporated, Addison-Wesley Publishing Company, Reading, Massachusetts, 1999. <https://adobe.com/jp/print/postscript/pdfs/PLRM.pdf>.
 - [2] A. Bayar. Towards an operational (I^A)T_EX package supporting optical scaling of dynamic mathematical symbols. *TUGboat* 37(2):171–179, 2016. <https://tug.org/TUGboat/tb37-2/tb116bayar.pdf>.
 - [3] A. Bayar. C¹ interpolation of sequences of points preserving convexity and obliquity based on oblique convex two-dimensional cubic Bézier splines. In *2024 IEEE International Conference on Signal, Image, Video and Communications (ISIVC 2024)*, pp. 1–6, Marrakech, Morocco, May 2024.
 - [4] A. Bayar. `dynMath`: A PostScript Type 3-based L^AT_EX package to support extensible mathematical symbols. *TUGboat* 45(1):18–24, 2024. <https://tug.org/TUGboat/tb45-1/tb139bayar-dynmath.pdf>
 - [5] D.E. Knuth. *The T_EXbook*, vol. A of *Computers and Typesetting*. Addison-Wesley, Reading, Massachusetts, 1st ed., 1984.
 - [6] L. Lamport. *L^AT_EX: A Document Preparation System*. Addison-Wesley, USA, 1994.
 - [7] LuaT_EX development team. *LuaT_EX Reference Manual*, Feb. 2024. <https://ctan.org/pkg/luatex>.
 - [8] T. Rokicki. *Dvips: A DVI-to-PostScript Translator*, Feb. 2024. <https://ctan.org/pkg/dvips>.
- ◇ Abdelouahad Bayar
Cadi Ayyad University—Higher School of Technology of Safi,
PSSII Lab
Sidi Aissa Road, PB 89
Safi, 46000
Morocco
a.bayar (at) uca dot ma
ORCID 0000-0002-3496-505X

TUG 2024 abstracts

Editor's note: Links to videos and other information are posted at tug.org/tug2024.

— * —

Vincent Goulet

A journey through the design of (yet another) journal class

Many scientific journals rely on document classes provided by large publishers. Yet, some journals still prefer to maintain their own class. The *Canadian Journal of Statistics | La revue canadienne de statistique* is one of them. In late 2022, I was commissioned to develop a class and bibliography styles that would update not only the production underpinnings, but also the visuals of the CJS. This presentation will provide a journey through the choices that I had to make along the way, the functionality that I imported from various sources, and the (hopefully) novel solutions I implemented.

Vincent Goulet

You (S)wave? Well, (S)tangle now!

The concept of literate programming, pioneered for TeX by Donald Knuth in the late 1970s, should be familiar to many in the (L)TeX community. Very briefly stated, it consists of a programming paradigm where a program and its documentation are interspersed in a single file. Source code and documentation are extracted respectively by `tangle` and `weave` procedures. Literate programming plays a central role nowadays in scientific computing for reproducible research purposes: instead of being hard coded into a report, results and graphics are woven in place using the computer code. In the R ecosystem, Sweave and knitr are widely used to build documents this way from R code. The aim of this presentation is to shed some light on the perhaps lesser-known component of literate programming, at least in scientific computing: the `tangle` step. I will describe a use case where a clever combination of weaving and tangling allows for efficiently maintaining a set of exercises and solutions.

Sarah Lang

LATEX in the Digital Humanities

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

Conference proceedings that require full paper submissions, exemplified by the Computational Humanities Conference, are becoming more prevalent in the Digital Humanities and thus, present a good reason to get acquainted with the necessary LATEX skills. This talk challenges the notion suggested by Quinn Dombrowski that learning LATEX is too much to ask of humanities scholars. Humanities scholars, even those with minimal technical background, can learn LATEX at the necessary level to format their submissions effectively, by using it essentially as a markup language.

The second and most important use case for LATEX in the Digital Humanities is in digital scholarly editing. There is a sizable digital scholarly editing community within the field. One could even say that digital scholarly editing is one of the core fields of work for digital humanists. Digital scholarly editing is a domain where the field initially flourished and continues to thrive, despite recent developments involving deep learning driven approaches and Large Language Model rendering the subfield of the "Computational Humanities" more and more dominant. Digital scholarly editing will continue to be a core task and technology within Digital Humanities; thus, LATEX will continue to remain relevant to the Digital Humanities.

Despite some early predictions that digital media would make physical books obsolete, we observe in 2024 that this transformation has not yet materialized and seems unlikely to occur in the near future. On the contrary, the value of the book as a material object and a symbol of cultural capital has surged, especially among younger audiences on platforms like BookTok. While this trend might seem irrelevant to academia per se, it underscores an important point: physical books retain their importance to this day and continue to be favored by textual scholars for various tasks, such as in-depth reading. This enduring preference for printed materials highlights the need of ensuring that digital humanities resources, such as digital scholarly editions, can be effectively translated into print when necessary. Transforming digital scholarly editions in TEI-XML format to LATEX allows us to produce print versions from digital editions, even when the primary intent of the edition is not to publish in print.

Through the transformation of TEI data using XSLT, potentially even employing large language models to generate LATEX code, scholars can create high-quality printed materials with little effort. This functionality is crucial for the digital humanities community, as it responds to the frequent preference for accessible, printable formats, which are particularly

useful in teaching settings. While digital editions offer unique benefits, like showcasing multiple witnesses or versions without prioritizing a single one, there remains a substantial demand for printed copies invaluable for detailed study, annotation, or instructional use. The `reledmac` package is a particularly useful tool for Digital Humanists.

The presentation also highlights the utility of L^AT_EX in managing complex documents within the humanities, such as archaeological catalogs. In fields like archaeology, it is commonplace to compile extensive catalogs of objects or findings as integral components of larger research projects, such as Ph.D. dissertations. Such catalogs often contain numerous images and can become unwieldy when managed with standard word processing software like Microsoft Word due to their size. Although this application is not exclusively within the digital humanities, it underscores the relevance of L^AT_EX across broader humanities disciplines, as has been discussed previously on the L^AT_EX Ninja blog.

Frank Mittelbach

Hooks, sockets and plugs

Driven by the need to support tagging, a number of ideas are being introduced into the L^AT_EX kernel to allow more flexible changes to code paths, design aspects and document command creation. Hooks were introduced a few releases ago, and provide a way to manage the interaction between packages in a flexible and powerful way. More recently, we brought in sockets and plugs: places where exactly one code path is needed, but what that code path is needs to be swappable. In this talk, we'll look at why we need both sockets and hooks, and which to use when.

Wim Obbels, Bart Snapp, Jim Fowler

Ximera interactive math educational resources for all: From L^AT_EX source code to PDF, HTML and beyond

Ximera is an open-source platform to create online interactive STEM courses using L^AT_EX as the source code. Initially developed at the Ohio State University (OSU), now around a dozen institutions are using Ximera materials. Backend development of the project has extended from just OSU to now include educators at the University of Florida and KU Leuven.

In this talk we will explain the basic ideas of how Ximera works and demonstrate how the currently available courses easily implement interactive questions, integrate applets from youtube, desmos or geogebra, hyperlinks, etc., and how this, based on T_EX4ht, rather smoothly translates between HTML and PDF.

After installing the Ximera package from CTAN, one can immediately generate PDF versions. To publish an online version an extra build environment (`xake`) is needed which enables publishing courses to a public server at `ximera.osu.edu` or optionally to a self-hosted server. Ximera can be integrated with learning management systems, collects useful learning analytics, and can return results to a gradebook.

It is currently under active development, getting extra functionality, more examples and tutorials, options for styling courses both online and in PDF, and integrated gradebook functionality. A docker setup will soon be available for easy deployment, and even a serverless setup with compilation-in-the-browser.

The CI/CD integration currently used at KU Leuven will be explained, which automatically updates the online content and all the corresponding PDFs whenever authorized authors git-push changes.

Interested users can learn more at github.com/ximeraProject/ximeraFirstSteps.

Martin Ruckert

The color concept of HiT_EX

One of the first feature requests for HiT_EX was colored text. Looking at the existing packages supporting colored text, it appears that their design was influenced if not limited significantly by the features provided by the T_EX engine, like `\special`, and the features provided by the output format, namely PDF.

Since HiT_EX has its own engine, its own output format, and its own viewing applications, it was possible to start a new design from scratch without such limitations, but not without conflicting design objectives. The new design should be simple, flexible, and powerful; it should be easy to understand and easy to implement; and it should be possible to support the existing color packages with only a limited implementation effort for a driver file.

It is planned to include the color support for HiT_EX in the 2025 edition of T_EX Live. I welcome feedback and suggestions for changes.

samcarter

The moloch beamer theme

The moloch beamer theme (ctan.org/pkg/moloch) by Johan Larsson is an updated fork of the well-known `metropolis` theme, which was originally written by Matthias Vogelgesang. In this short talk, I'm going to quickly introduce the theme and then show, from the perspective of a user, how to switch from the `metropolis` theme to the new `moloch` theme.

Tyge Tiessen

Rewriting T_EX today

T_EX82 was written forty years ago. The constrained memory resources as well as portability concerns

and compiler limitations of that time have set strict bounds to the original implementation. As some consequences of these bounds TeX82 features handwritten dynamic memory management, ubiquitous use of global variables, and manual string management.

In this talk, I will discuss the joys and challenges of rewriting TeX82 today without these limitations.

Jan Vaněk, Hân Thé Thành

Exploring Primo: A developer's perspective

In this session, we'll take a practical look at Primo from the viewpoint of developers. We'll examine its technology stack, code structure, and document model, providing insights into its implementation for collaboration.

We'll delve into the front-end development aspects, covering the usage of the VDL JavaScript library for UI components and the related CSS code. Additionally, we'll discuss the deployment process and touch upon the importance of TeX compilation and XML validation in the Primo system.

Didier Verna

A couple of extensions to the Knuth-Plass algorithm

In this talk, we will present our implementation of a couple of extensions to the Knuth-Plass algorithm in ETAP, our experimental typesetting algorithms platform.

Joseph Wright

siunitx development continues: 2024

The `siunitx` package was first released in 2008, and has been through three major revisions; v3.0.0 was released in May 2021. Since then, development of new ideas has continued, with new features in numbers, tables and units. Here, I will pick out some highlights from the past three years of work, and look at where we might see additional ideas in the future.

Joseph Wright

Templates: Prototype document elements

Controlling design is something that has been challenging in L^AT_EX to date. While the L^AT_EX Team developed experimental ideas in the mid-1990s for creating flexible design elements, they were not viable for real documents then. These ideas, based around ‘templates’, have now reached maturity and are included in the Summer 2024 L^AT_EX kernel release.

In this talk, I will look at what a template is, why we'd want to use them and the flexibility and power they will bring to controlling document design.

Die T_EXnische Komödie 2/2024

Die T_EXnische Komödie is the journal of DANTE e.V., the German-language TeX user group (dante.de).

Die T_EXnische Komödie 2/2024

MARTIN SIEVERS, Grußwort [Greeting]; pp. 4–5
Introductory words from the DANTE president.

VOLKER RW SCHAA, Protokoll der 66.
Mitgliederversammlung von DANTE e.V. im
Goethe-Nationalmuseum in Weimar [Minutes of
the 66th general meeting of DANTE e.V. in the
Goethe National Museum in Weimar]; pp. 6–12
The official minutes of the general meeting.

HENRIK GASMUS, Die diesjährige Frühjahrstagung
von DANTE e.V. vom 4.–6. April 2024 im
Goethe-Nationalmuseum in Weimar [This year's
spring conference of DANTE e.V. from 4th to 6th
April 2024 in the Goethe National Museum in
Weimar]; pp. 13–24

A report on the spring convention in the Goethe-
museum in Weimar.

TEAM STAND DANTE E.V., Chemnitzer
Linux-Tage 24 [Chemnitz Linux-Days 24];
pp. 24–25

Some impressions of the DANTE booth at the
Chemnitz Linux days.

KENO WEHR, L^AT_EX und Schulphysik 6:
Feldlinienbilder [L^AT_EX and School Physics 6:
Field line images]; pp. 27–51

A new article in the school physics series, this
time on how to draw images of lines of force in a
field.

RALF MISPELHORN, Umgang mit Bildern
[Handling images]; pp. 51–58

A comprehensive summary on how to handle
images in L^AT_EX.

ROLF NIEPRASCHK, Von Markdown zu PDF
[From Markdown to PDF]; pp. 59–64

On the conversion of Markdown to PDF.

HENNING HRABAN RAMM, ConTeXt kurz notiert
[ConTeXt short notes]; pp. 64–68

Some news from the ConTeXt world.

JÜRGEN FENN, Neue Pakete auf CTAN [New
packages on CTAN]; pp. 69–73

[Received from Uwe Ziegenhagen.]

La Lettre GUTenberg 52, 2024

La Lettre GUTenberg is a publication of GUTenberg, the French-language TeX user group (gutenberg-asso.org)

La Lettre GUTenberg #52

Published April 30, 2024.
doi.org/10.60028/lettre.vi52

PATRICK BIDEAULT, Éditorial [Editorial]; pp. 1–4

MAXIME CHUPIN & BASTIEN DUMONT, Compte rendu de la Journée GUTenberg du 18 novembre 2023 [Report of GUTenberg Day 2023]; pp. 5–9

MAXIME CHUPIN, Compte rendu de l'assemblée générale du 18 novembre 2023 [Report of the General Assembly 2023]; pp. 9–11

PATRICK BIDEAULT & DENIS BITOUZÉ, Compte rendu de la réunion du conseil d'administration du dimanche 4 février 2024 [Report of the board's meetings]; pp. 12–15

MAXIME CHUPIN & PATRICK BIDEAULT, Exposés mensuels sur (L)TeX et autres logiciels [Monthly conferences]; pp. 16–19

Short reports about the monthly online conferences that GUTenberg offers.

DENIS BITOUZÉ, Petite histoire de la FAQ L^ATeX GUTenberg [A short history of L^ATeX GUTenberg's FAQ]; pp. 20–32

History of the French FAQ, by Denis Bitouzé.

DENIS BITOUZÉ, Fonctionnement de la FAQ L^ATeX GUTenberg [Detailed presentation of GUTenberg's L^ATeX FAQ]; pp. 33–46

The French FAQ has recently been renewed. It is now an online collaborative service that's constantly updated.

PATRICK BIDEAULT, MAXIME CHUPIN & BASTIEN DUMONT, Et maintenant, une bonne *vieille* veille technologique ! [Technology watch]; pp. 46–57

November 2023–March 2024: 80 new CTAN packages, including 12 from French-speaking countries.

MAXIME CHUPIN, LuaL^ATeX & MetaPost au service de la vulgarisation [LuaL^ATeX & MetaPost at the service of popularisation]; pp. 58–68

How the above-mentioned software helped the author explain his scientific activity to secondary school pupils.

MAREI PEISCHL, \dante_tutorial:nn{expl3} {2022} [An introduction to expl3]; pp. 69–83

A translation of the article published in *Die T_EXnische Komödie* 2022/4 and in *TUGboat* 44:1, 2023.

MAXIME CHUPIN, Pas à pas : installation, via le réseau, de la T_EX Live 2024 [Step by step: Net install of T_EX Live 2024]; pp. 84–89

JACQUES ANDRÉ, Feuille aldine ou... grappe ? [Floral heart or... grapes?]; pp. 90–100

Aldus Manutius' floral heart or Erhard Ratdolt's bunch of grapes?

JACQUES ANDRÉ, PATRICK BIDEAULT & MAXIME CHUPIN, La fonte de ce numéro : Luciole [This issue's font: Luciole]; pp. 100–107

The Luciole font has been designed specifically for visually impaired people. This publicly funded project is the result of more than two years of scientific collaboration between scientists and designers.

PATRICK BIDEAULT & JACQUES ANDRÉ, Comptes rendus de lecture [Book reviews]; pp. 107–109

About Graphisme en France #29, now published also in English (www.cnap.fr/en/publishing/graphisme-en-france-issue-29) and about Marc Smith's True Story of the At Sign.

PATRICK BIDEAULT & MAXIME CHUPIN, En bref [At a glance]; pp. 109–111

Short news about Glenn Fleishman's "London Kerning" and GUTenberg at the Open Educational Software Day.

YVON HENEL, Rébus [A rebus]; p. 111

Prochaines rencontres [The next T_EX-related meetings in Europe]; pp. 111–112

[Received from Patrick Bideault.]

MAPS 53–54 (2023–2024)

MAPS is the publication of NTG, the Dutch language *TEX* user group (<https://ntg.nl>).

MAPS 53 (Spring 2023)

REDACTIONEEL, Editorial; pp. 1–2

This is a special edition of *MAPS* containing five articles by Taco Hoekwater about various fundamental matters in MetaPost.

TACO HOEKWATER, Introduction; pp. 3–4

What you have here is a series of articles about details of the MetaPost programming language.

The target audience of these articles are users that are already somewhat familiar with simple graphics in MetaPost but want to have a clearer understanding of the language to make better use of its possibilities.

Each of the articles discusses a specific subsystem and together they should provide a solid base for improving the reader's knowledge of MetaPost.

TACO HOEKWATER, Variables: Sparks, tags, suffixes and subscripts; pp. 5–18

MetaPost variables are rather complicated things. This article will attempt to explain the various uses of type declarations, saves, and vardefs.

TACO HOEKWATER, Definitions; pp. 19–34

Definitions in MetaPost are also a fairly complicated subject. This article tries to cover everything you need to know about writing your own definitions, but it assumes a fair bit of familiarity with MetaPost's data types and general syntax. In particular, I assume you have read the preceding "Sparks, tags, suffixes and subscripts" article.

TACO HOEKWATER, Paths, pairs, pens and transforms; pp. 35–72

This article 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 the "Sparks, tags, suffixes and subscripts" article.

I will first discuss the creating of paths, followed by the creating of pairs, and then the creating of pens. Finally, I will discuss the operations on those items, for instance, by using transformations.

TACO HOEKWATER, Conditionals and loops; pp. 73–80

This article is about how to make your program decide what to do next: conditions and loops.

TACO HOEKWATER, Colors and pictures; pp. 80–96

This article is about MetaPost output. MetaPost produces graphics by means of picture variables that can contain a few different object types. The most important drawing object types can be colorized, so the first part of this article will talk about color data structures.

MAPS 54 (Spring 2024)

MAPS REDACTIE, Redactieel [Editorial]; p. 1

SANDER VAN GELOVEN, Afbreekpatronen [Hyphenation patterns]; p. 2

BOB WITMAN, Het eeuwige leven [Eternal life]; pp. 3–4
‘Letterman’ Middendorp wrote the standard work on Dutch typography, born from boundless curiosity.

WILLI EGGER, Kaktovik nummers met basis 20 [Kaktovik numbers with base 20]; pp. 5–8

We are used to navigating the Arabic number system using base 10. However, it is also possible to choose a different base. The Inuit, for example, use base 20 to count. This article discusses the number system of these Inuit.

Y. ROBBERS, Tante Lenie weet raad... [Aunt Lenie knows advice...]; pp. 9–14

Spring has begun, Valentine's Day is over, and love troubles make way for *TEX* problems. And for both, you can always turn to your Aunt Lenie! This time she helps Lisa G. create an attractive brochure for her cat café using *LATEX*, assists student Jenia G. with her font problem using *LATEX3*, helps Machteld K. with Greek counters in *LATEX*, uses PGF/TikZ to help Hans M. with new floats for his new syllabus, and addresses two new problems from Herman R., who is still doing complicated mathematical things in plain *TEX*.

ERIK NIJENHUIS, Documenten in YAML specificeren en invullen met lua-placeholders in *LATEX* [Specifying documents in YAML and filling them with lua-placeholders in *LATEX*]; pp. 15–27

[Published in English in *TUGboat* 45:1.]

DUSTIN HENDRIKS, Verkenning van het automatiseren van *LATEX* met programmatuur [Exploring the automation of *LATEX* with software]; pp. 28–30

In this article, I share my experiences and insights in programming the software ELDYN. This tool enables file manipulation by dynamically injecting variables and templates. This can provide benefits in the automated creation of strictly structured documents, particularly through *LATEX*. In this article, I share background information about myself, share my insights in programming the mentioned tool, and hope to inspire by sharing my demonstration project WebTeX: a concrete application of the developed software.

DENIS MAIER, Automatic suppression of unwanted ligatures when typesetting German; pp. 31–35

An approach using ConTeXt LMTX's language options.

PIETER VAN OOSTRUM, Software Engineering; pp. 36–44

This article describes some (software engineering) practices that I used to develop the *fancyhdr* *LATEX* package. The practices are very general, however, and certainly not exclusive for *LATEX* packages.

HANS HAGEN, Debugging; pp. 45–56

Verbose logging and more visual features. The ConTeXt ‘lowlevel’ manuals have more details.

HANS HAGEN, Lua in TeX; pp. 57–62

At the end of 2023 the Lua language celebrated its 30th anniversary and in that perspective I offered to wrap up our experiences with that language from the perspective of TeX. This wrapup is not aimed at TeX users, but can nevertheless give them some background and a status overview.

HANS HAGEN, MIKAEL SUNDQVIST, Meaningful math; pp. 63–74

In this article we’re going to discuss math from the perspective of accessibility. Although ConTeXt has already supported tagging in PDF for quite a while, that specific kind of accessibility never took off, if only because very few viewers did anything useful with it. However, with universities introducing (whatever) validating features for documents pushed into the systems used for teaching, there was no way to avoid picking up this thread.

We start with some reflections about how we got here and then move on to some examples of how we deal with this in LMTX. This project is part of a larger effort to get even better typeset math out of TeX so we could benefit from some new features in the engine, even if they were not added with accessibility in mind.

CLAUDIO BECCARI, Albanian hyphenation; pp. 75–83

After a short historical review of the Albanian language, the procedure used to create the Albanian hyphenation pattern file is described.

BENJAMIN WACHE, Writing a PhD thesis in L^AT_EX; pp. 84–86

In this article I explain the setup I used for editing my PhD thesis using L^AT_EX, including use of Overleaf and ChatGPT.

MIKAEL SUNDQVIST, Making a simple photo book with ConTeXt; pp. 87–93

In Spring 2022 a question entitled “How to achieve a few different page layouts for a photobook?” appeared on Stack Exchange (<https://tex.stackexchange.com/q/643009>). User ‘ana’ asked how one in L^AT_EX could setup a document with layout requirements for a typical photobook. Since I am interested in both photography and typesetting, I thought it would be fun to provide an answer, even though it would involve using the “wrong” tool, ConTeXt.

HANS HAGEN, How not to install ConTeXt; p. 94

An annotated review of ChatGPT’s procedure for installing ConTeXt.

FABRICE LARRIBE, MetaFun for movies; pp. 95–106

This article shows how MetaFun can be used to make movies, by showing the construction of several projects, step by step.

[Received from Wybo Dekker.]

TUG bursary committee report for 2024

Jim Hefferon

The TeX Users Group Bursary Fund provides grants to help members of our community who would like to attend the annual TUG Conference but who would find it a financial hardship. This is the report of the Bursary Committee’s 2024 results, along with some general background on the bursary.

Background

We encourage everyone interested in attending the annual conference but who is looking for financial assistance to apply. In particular, we encourage applicants from traditionally underrepresented groups and underrepresented areas. An applicant need not be a TUG member.

This is our first annual report to appear in *TUGboat* (we intend to publish future reports here). Because of that, we will briefly describe our deliberations. Of course, TUG has limited resources so the amount available is limited and the committee may need to judge among applications. We consider whether the applicant will give a presentation or be a panel participant, as well as whether they will write an article for the conference’s *TUGboat* issue. Other positive factors in an application are that the person is able to attend the entire conference, including being available for informal interactions, and is a member of a TeX-related user group. That said, however, the committee aims to help all applicants wherever possible. We’ve found that in recent years the number of applications is not large, so again we encourage interested people to apply.

Note that while TUG provides administrative support for these funds, the Bursary Committee operates independently and the TUG Board is not given individually identifying knowledge of applicants.

Finally, we ask all members of the community to please consider making a donation to the Bursary Fund. All monies are used for the stated purpose; there are no administrative charges and if there is a surplus remaining after disbursements then it is carried over to the next year.

In 2024

The Bursary Committee this year was Jim Hefferon (Chair), Carla Maggi, and Karl Berry (TUG Treasurer, ex officio). The application deadline was May 1, 2024. There was a grant to one individual for about \$1400. (This is only an estimate because there are sometimes small last-minute cost changes.)

For more information, see tug.org/bursary/. That site has summary reports back to 2003 (there

was no bursary during the COVID years 2020–2022 as the conferences were online and free of charge). The exact grant amount will be posted there after the conference. In addition, every year we post a link to the application for the next year's conference, once its details are settled.

- ◊ Jim Hefferon
Mathematics and Statistics
University of Vermont
`tug-bursary (at) tug dot org`
<https://tug.org/bursary>

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 L^AT_EX consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom L^AT_EX packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in TeX and three decades of experience in teaching & training. I have authored more than forty packages on CTAN as well as Perl packages on CPAN and R packages on CRAN, published papers in TeX-related journals, and conducted several workshops on TeX and related subjects. Among my customers have been Amnesty International, Annals of Mathematics, ACM, FAO UN, Google, Israel Journal of Mathematics, No Starch Press, Philosophers' Imprint, Res Philosophica, US Army Corps of Engineers, US Treasury, and many others.

We recently expanded our staff and operations to provide copy-editing, cleaning and troubleshooting of TeX manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

Dangerous Curve

Email: `khargreaves (at) gmail.com`

Typesetting for over 40 years, we have experience in production typography, graphic design, font design, and computer science, to name a few things. One DC co-owner co-authored, designed, and illustrated a TeX book (*TeX for the Impatient*).

We can:

- convert your documents to L^AT_EX 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

100 Centre Street #420
Brookline, MA 02446
+1 617-823-9938
Email: `amyh (at) texnology.com`
Web: www.texnology.com

Full time L^AT_EX 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.

- *L^AT_EX Macro Writing*: Packages for books, journals, slides, posters, e-publishing and more; Sophisticated documentation for users.
- Data Visualization, database publishing.
- Innovative uses for L^AT_EX, creative solutions our speciality.
- L^AT_EX Training, customized to your needs, on-site or via Zoom. See

Henrickson, Amy (cont'd)

<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](mailto:david.latchman(at)texnical-designs.com)

Web: www.texnical-designs.com

L^AT_EX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized L^AT_EX packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

L^AT_EX Typesetting

Boston, MA

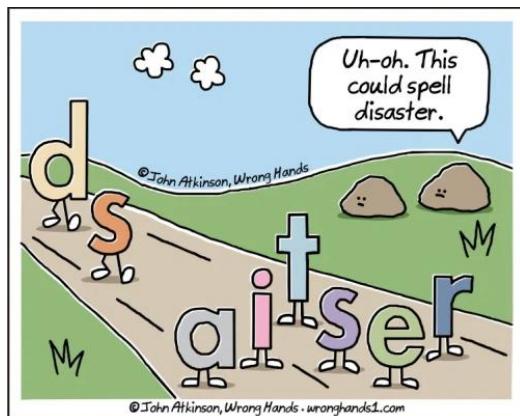
Email: [enquiries \(at\) latextypesetting.com](mailto:enquiries(at)latextypesetting.com)

Web: www.latextypesetting.com

L^AT_EX 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 L^AT_EX 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 L^AT_EX. 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 L^AT_EX and CSS; production of books and journals for any layout with publish-ready PDF, HTML and XML from L^AT_EX (bypassing any publishers' processes); custom development of L^AT_EX packages with documentation; copyediting and proofreading for English; training in L^AT_EX for authors, publishers and typesetters.

I have over two decades of experience in academic publishing, helping authors, publishers and typesetters use L^AT_EX. I've built typesetting and conversion systems with L^AT_EX and provided T_EX support for a major publisher.

Warde, Jake

90 Resaca Ave.

Box 452

Forest Knolls, CA 94933

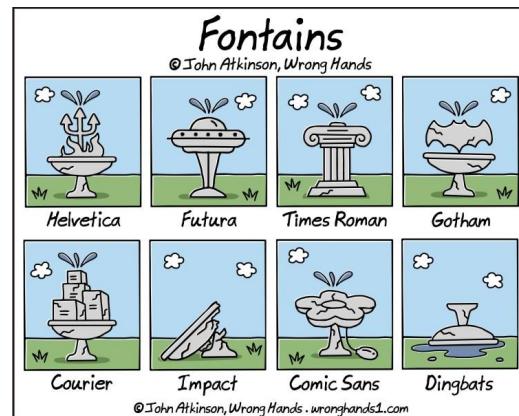
+1 650-468-1393

Email: [jwarde \(at\) wardepub.com](mailto:jwarde(at)wardepub.com)

Web: myprojectnotebook.com

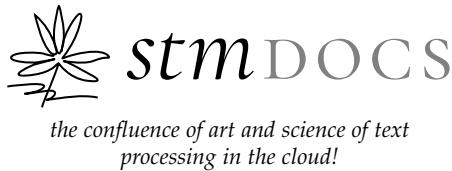
I have been in academic publishing for 30+ years. I was a linguistics major at Stanford in the mid-1970s, then started a publishing career. I knew about T_EX from editors at Addison-Wesley who were using it to publish beautifully set math and computer science books.

Long story short, I started using L^AT_EX 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 T_EX, I can help. And if I cannot help I'll let you know right away.



Science is what we understand well enough to explain to a computer. Art is everything else we do.

— Donald E. Knuth



- empowering authors to self-publish
- assisted authoring
- TeXFolio — the complete journal production in the cloud
- NEPTUNE — proofing framework for TeX authors

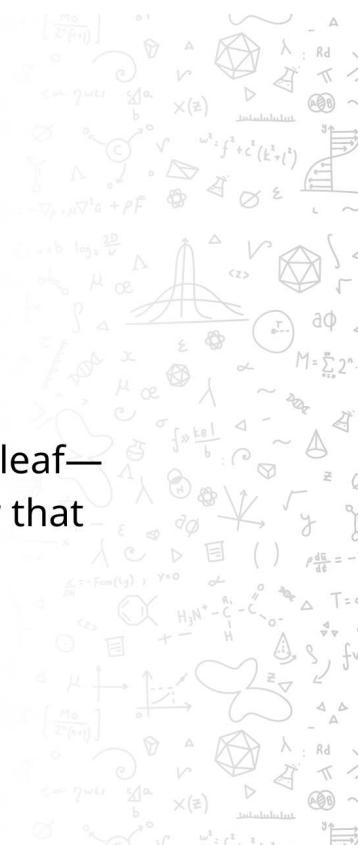
STM DOCUMENT ENGINEERING PVT LTD
Trivandrum • India 695571 • www.stmdocs.in • info@stmdocs.in



\begin{anything}

Write like a rocket scientist with Overleaf—the collaborative, online LaTeX editor that anyone can use.

www.overleaf.com



Calendar

2024

- Aug 20–23 24th ACM Symposium on Document Engineering, Adobe, San Jose, California. doceng.org/doceng2024
- Sep 3–13 arXiv Accessibility Forum 2024 (fully remote). accessibility2024.arxiv.org
- Sep 10 Type Tuesday: My (favourite) Font, St Bride Foundation, London, England. sbf.org.uk/whats-on
- Sep 15–20 XML Summer School, St Edmund Hall, Oxford University, Oxford, UK. xmlsummerschool.com
- Oct 4 *TUGboat* 45:3, submission deadline.
- Oct 6 Ladies of Letterpress, Letterpress Summit 2024, “State of the art and craft” (online), ladiesofletterpress.com/conference
- Oct 12 GuIT Meeting 2024, Brescia, Italy. www.guitex.org/home/en/meeting
- Oct 17 The Beatrice Warde Memorial Lecture, St Bride Foundation, London, England. sbf.org.uk/whats-on
- Oct 23–25 Grapholinguistics in the 21st century—From graphemes to knowledge, (Donald Knuth is listed as a presenter) Università Ca' Foscari, Venice, Italy. grafematik2024.sciencesconf.org
- Oct 31 Tour of St Bride Foundation, London, England. sbf.org.uk/whats-on

Dec 3–6

- SIGGRAPH Asia 2024, “Curious Minds”, Tokyo International Forum, Tokyo, Japan. asia.siggraph.org/2024

2025

- Mar 1 **TUG election:** nominations due, 07:00 a.m. PST. tug.org/election
- Mar 6–8 TypoDay2025, “Typography and Storytelling”, IDC School of Design, Indian Institute of Technology Bombay, Bombay, India. www.typoday.in
- Mar 21 *TUGboat* 46:1, submission deadline.
- Apr 22–26 Association Typographique Internationale, ATypI Copenhagen 2025 Copenhagen, Denmark. atypi.org/conferences-events/atypi-copenhagen-2025
- Jun 25–27 Twenty-third International Conference on New Directions in the Humanities, “Oceanic Journeys: Multicultural Approaches in the Humanities”, University of Hawaii, Hilo, Hawaii (and online). thehumanities.com/2025-conference
- Jul 15–18 Digital Humanities 2025, Alliance of Digital Humanities Organizations, Universidade NOVA de Lisboa, Lisbon, Portugal. adho.org/conference
- Sep 5 The Updike Prize for Student Type Design, application deadline, 5:00 p.m. EST. Providence Public Library, Providence, Rhode Island. prov.pub/updikeprize

Status as of 20 August 2024

For additional information on TUG-sponsored events listed here, contact the TUG office by email: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

User group meeting announcements are posted at tug.org/meetings. Interested users can subscribe and/or post to the related mailing list, and are encouraged to do so.

Other calendars of typographic interest are linked from tug.org/calendar.