TUGboat

Volume 41, Number 2 / 2020
TUG 2020 Conference Proceedings

TEX Users Group

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

Individual memberships

2020 dues for individual members are as follows:

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

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

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

Journal subscriptions

TUGboat subscriptions (non-voting) are available to libraries and other organizations or individuals for whom memberships are either not appropriate or desired. Subscriptions are delivered on a calendar year basis. The subscription rate for 2020 is $110.

Institutional memberships

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

Trademarks

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

[printing date: September 2020]

Printed in U.S.A.

Addresses

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

Telephone

+1 503 223-9994

Fax

+1 815 301-3568

Web

`tug.org`
`tug.org/TUGboat`

Electronic Mail

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

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

Technical support for TEX users:
`support@tug.org`

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

2020 Conference Proceedings

TEX Users Group
Forty-first annual TUG conference
Online
July 24–26, 2020

TUGBOAT

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

Volume 41, Number 2, 2020
Portland, Oregon, U.S.A.

TUG 2020 — Online — July 26–28, 2020

The forty-first annual TUG conference
https://tug.org/2020 ▪ tug2020@tug.org

Conference committee

Karl Berry
Jennifer Claudio
Rohit Goswami
Robin Laakso
Ross Moore
Norbert Preining
Will Robertson
Paulo Ney de Souza, principal organizer
Boris Veytsman

Thanks to all!

Sponsors

TEX Users Group
DANTE e.V.
Adobe Inc.
Cary Graphic Arts Collection
Overleaf
STM Document Engineering Pvt Ltd
The University of Adelaide
with generous assistance from many individual contributors.

TUG 2020 program

(All times and days here are PDT = UTC−7.) (* = presenter)

Thursday July 23	09:00–	Cheryl Ponchin*, IDA/CCR-P; Sue DeMeritt*, IDA/CCR-La Jolla	*Introductory LaTeX workshop*
Friday July 24	08:55	Paulo Ney de Souza	*Welcome*
	09:00	Steven Matteson, Monotype	*The road to Noto — TUG 2020 keynote address*
	10:00	Michael Sharpe, UC San Diego	*The newest changes to newtx and its relatives and codependents*
	11:30	Jennifer Claudio, San Jose, CA	*Typographical expression of emotions in a variety of alphabet systems*
	13:00	Amelia Hugill-Fontanel, Cary Graphic Arts Collection at RIT	*The creative evolution of type specimens*
	13:45	David Walden, E. Sandwich, MA	*Noticing history, a personal view*
	14:30	Paulo Cereda, Island of TEX	*The Island of TEX: Developing abroad — your next destination*
	15:15	Boris Veytsman, Chan Zuckerberg Initiative and George Mason Univ.	*Using Overleaf for collaborative projects: First impressions and lessons learned*
	16:00	Ross Moore, Macquarie University, Sydney, Australia	*CMaps, Virtual fonts, ActualText for reliable text extraction and accessibility*
	16:45	Paulo Ney de Souza*, UC Berkeley and BooksInBytes; Vadim Ponomarev, PetrSU	*dePSFrag, the final nail in the coffin*

TUG 2020 program (continued)

Saturday July 25	00:00	Astrid Schmölzer*, U. Bamberg; Sarah Lang*, Karl-Franzens-U. Graz	*Empowerment and teaching LaTeX*
	00:45	Jonathan Fine	*Learning LaTeX (and other languages) online*
	01:30	Joseph Wright, LaTeX Project	*learnlatex.org: Taking online training LaTeX fully interactive*
	02:15	samcarter	*TopTeX, a new Q&A site for TeX*
	03:45	Patrick Gundlach, speedata, Germany	*Speedata Publisher — a different approach to typesetting using LuaTeX*
	04:30	Peter Flynn, Silmaril Consultants	*Your personal LaTeX bookshelf: Improving your background in a time of lockdown*
	05:15	Frank Mittelbach, LaTeX Project	*Quo vadis LaTeX(3) Team — A look back and at the upcoming years*
	06:45	Rajeesh KV*, Aravind Rajendran, STM Document Engineering	*Beyond Roman fonts: Extra dimensions in Malayalam fonts*
	07:30	Marcel Krüger, LaTeX Project	*HarfBuzz in LuaLaTeX*
	08:15	Paulo Ney de Souza	*Interview with Javier Bezos*
	09:45	Sarah Lang	*Didactical reduction versus references. How to better teach LaTeX*
	10:30	Jim Hefferon, St. Michael's Coll.	*A first set of LaTeX packages*
	11:15	Paul Gessler, Overleaf	*Teaching with LaTeX and Overleaf*
	12:00	Paulo Ney de Souza	*Interview with Philip Kime*
	21:00	Rishi T*, Aravind Rajendran, STM Document Engineering	*LaTeX technologies at work — aesthetically beautiful PDFs on the fly from XML input: XML page composition (XPC) microservice in the cloud*
	21:45	Takuto Asakura, U. of Tokyo	*The design concept for llmk — Light LaTeX Make*
	22:30	Norbert Preining, Accelia, TeX Live	*TeX Live 2020 news;* `texlive.info` *services*
Sunday July 26	06:00	Eric Mc Sween	*Making a new TeX Live release available on Overleaf*
	06:45	Yoan Tournade, YtoTech, France	*Latex-on-HTTP: LaTeX as a cloud document edition service for webapp developers*
	07:30	Jonathan Fine	*TeX and LaTeX: The user experience*
	08:15	James Carlson	*Interactive compilation of LaTeX for web browsers*
	09:45	Martin Ruckert, Munich University of Applied Sciences	*The HINT project: Status and open questions*
	10:30	Marcel Krüger	*MetaPost-based, dynamic extensible delimiters for LuaTeX*
	11:15	Brandon Rhodes, RhodesMill.org	*Typesetting with Python*
	12:00	Patrick Ion	*TeX and global mathematics*
	13:30	William Hammond, San Diego	*Why the LaTeX community should care about SGML*
	14:15	John MacFarlane, Department of Philosophy, UC Berkeley	*Pandoc for TeXnicians — TUG 2020 keynote address*
	≈15:15	*end*	

Random musings on TUG 2020 online

Barbara Beeton

It happened! It worked!

The coronavirus pandemic threw a monkey wrench in the plans to hold TUG 2020 at the Rochester Institute of Technology. So a decision to hold it online was inevitable, if the conference was to be held at all. With the help of many people, everything came together and an exciting program was presented with remarkably few glitches.

First of all, thanks to everyone who worked so hard to make the conference a success. Will Robertson obtained permission from his institution, the University of Adelaide, for use of their Zoom license. Paulo Ney de Souza's experience with online presentations was invaluable in putting together the technical setup; Paulo was also the most visible host during the program, introducing presentations and conducting two live interviews. Other hosts included Arthur Rosendahl (né Reutenauer), Ross Moore, Norbert Preining, Tom Hejda, and probably others while I was asleep. Jennifer Claudio produced the poster, and helped in many ways with the new online world. Paulo Cereda created the attendance certificates and provided feedback on many fronts, as well as his usual much-needed good cheer. Thanks to all!

Arrangement of the schedule was done largely by Karl Berry, with consideration given to the time zone inhabited by each speaker, so that their presence would be slotted in a "comfortable" time, not when they would normally be asleep. Since speakers were located in almost all parts of the world, with the largest gap being the watery expanse of the Pacific Ocean, the relevant time zone was requested at the time of registration. The schedule as presented to potential viewers was tailored to give times in *their* local time zone. (This feature also worked for some who hadn't registered, as my husband determined when he added the schedule to his calendar so he would know when he could interrupt me for dinner.)

Most speakers made videos of their presentations. This provided some insurance against unforeseen scheduling problems (in the event, only one speaker was unable to be present at the scheduled time), and also makes it relatively straightforward to reprocess what was seen by online viewers, for a permanent presence on YouTube. Reprocessing is expected to take some time, perhaps a few weeks, but, with luck, it should be possible to watch via YouTube links by the time the proceedings are published.

In addition to the primary Zoom feed, the entire conference (including "dead" hours) was streamed to YouTube, and I believe there was another secondary feed. (Using a new laptop, not fully configured, I wasn't able to connect via Zoom, but watched the YouTube feed, and was thus unable to post interactive questions or participate in the apparently active chat.) Chat rooms were set up (on Zulip and Gather Town) for asking questions during talks as well as for social interaction; this was managed by Rohit Goswami in Iceland.

The backgrounds shown in most speakers' videos were either obviously book-rich or rather minimal — this latter being recommended for video presentation. But one background stood out: Dave Walden sitting on his porch with his back to an enticing marsh. Unfair to those of us sweltering at 30+ °C with humidity.

The wide geographic distribution was made real in views of two of the hosts, during setup and shutdown time: Ross, wrapped in a hoodie, commenting that it's cold during the Australian winter! And Arthur, saying goodnight in Sweden at local time heading on to midnight, with bright sun still streaming in the window.

I heard after the fact that one of the talks had been Zoom-bombed. Although a disruption at the time, the talk had been pre-recorded, so later viewers won't have to suffer the indignity. Oh, for the days when politeness was the norm, and not a rarity.

When the videos are posted, I shall first watch the talks that I missed — too many of them that I really wanted to see but just couldn't stay awake. Then I shall watch the others again, starting with the virtual tour of the Cary Library at RIT. (It's posted at `https://youtu.be/7Cm2AcQiUuk`.) I've been there before, but it's a little different every time, and I've never had a fully guided tour; Amelia Hugill-Fontanel clearly loves her charges, and I'm hoping that we're free enough of this virus by next summer that we can visit in person. For the rest, there's something interesting to be learned from every talk, so I won't pick favorites here.

Finally, an unintentional feature appearing during the live interludes between talks was the presence of cats insinuating themselves into camera range with many of the speakers and hosts; cats have their own imperatives, namely curiosity and being "in the way". I think I'll miss that when we can all get together in person again.

◇ Barbara Beeton
`https://tug.org/TUGboat`
`tugboat (at) tug dot org`

Observations on the TEX Users Group's 41st Annual Conference — TUG 2020 in the COVID-19 era

David Walden

The 41st annual conference of the TEX Users Group (TUG, `tug.org`) was scheduled to be at the Cary Graphic Arts Collection of the Rochester Institute of Technology this past summer.

Like so many other organizations in 2020, TUG's board of directors had to cancel its in-person conference on account of the 2020 COVID-19 epidemic. After a short period of indecision, the board decided to try to have a Zoom-based conference, and they asked TUG member Paulo Ney de Souza to organize the online conference. Paulo had substantial prior experience organizing online conferences in response to cancellation of in-person conferences.

I am partially writing this report for people outside the TUG community and *TUGboat* readers who may be interested in how organizations handled moving from an in-person to an online conference. Of course, it is also written for *TUGboat* readers who attended or may have missed the conference.

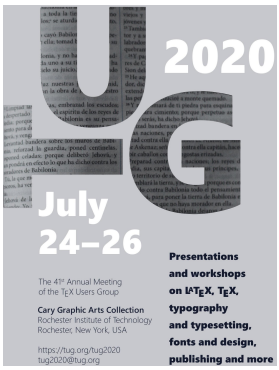


Figure 1: Left: Poster for original conference, designed by Maggie Blaisdell, an RIT graphic design student. Right: for the online conference, designed by Jennifer Claudio, a science teacher in San Jose.

The event

The virtual conference was held July 24–26, the same days for which the in-person conference had been scheduled; people already had those days blocked out in their calendars. In partnership with the University of Adelaide, where TUG board member Will Robertson is on the faculty and made the arrangements, the conference was broadcast on Zoom. The conference was also streamed on YouTube because it was easier for some people to watch via YouTube than Zoom; it also provided a back-up access path.

Conference registration was required, free (unlike TUG in-person conferences), but with encouragement to contribute to TUG. About a seventh of registered attendees did contribute, as did several institutions.

The online conference worked as follows:

1. Conference presenters were encouraged to record their presentations on video in advance and to pass them to the conference organizers for broadcast during the presenter's slot in the conference schedule. Participants were given excellent information about putting a presentation on video, available at `tug.org/tug2020/pres.html`. After the presentation was shown, the presenter was available live to answer questions.
2. On July 23 (the day before the main conference talks started), an online introductory LATEX workshop was held; Overleaf generously provided support for online TEX usage at the workshop. The workshop leaders, Sue DeMeritt and Cheryl Ponchin, had held similar workshops in person at many previous TUG conferences. About 30 people joined for more than half of the workshop; many others were present for shorter amounts of time.
3. The daily conference schedule was organized so presenters could give their talks at a reasonable time within their own time zone (`tug.org/tug2020/program.html`). The conference was run more or less around the clock; times here are US EDT: day 1, Friday noon to 8:30pm; day 2, Saturday, 3am to 4pm and 9:45pm to 3am Sunday; day 3, Sunday, 9am to 7pm.
4. For every session, there were declared hosts or co-hosts with Paulo being the primary host when he was not taking a break to sleep. The hosts moderated the sessions.
5. During the sessions about a dozen and a half people — usually other speakers, but occasionally attendees — were designated to be "panelists" within Zoom. Panelists could unmute and speak (as the hosts could); the rest of the people watching a session could not unmute. Everyone participating in a session could send questions and chats (in a text box) which everyone on Zoom could see.
6. Presenters' videos or slides resided with multiple hosts so, when they didn't work from one host's location, they could be shared from another host's location. A few presenters showed their slides and gave their presentations live.
7. There were also virtual breakout rooms where people could meet for technical discussion or

socially, using Zulip (zulipchat.com, around 80 people over the course of the conference) and Gather Town (gather.town, up to 15–20 people). These were organized by Rohit Goswami of the University of Iceland. Both remain active indefinitely and anyone is welcome to join.

The content

There were presentations on quite a variety of topics. See again tug.org/tug2020/program.html. As is typical of TUG conferences, there are a number of presentations not closely related to TEX.

I saw the presentations as falling into a number of areas: creating accessible PDF output; TEX community infrastructure and resources and their use, including new and commercial ones; teaching TEX and the user experience; typography; applications of TEX et al.; connections between TEX and other systems and formats; extending use into other domains and to other devices; variations and improvements on traditional TEX capabilities; and the future of (LA)TEX, etc., both plans and opinions of where else to go. There were two keynote presentations and two live interviews.

There were lots of very fine presentations, and there are links to the videos of the presentations at tug.org/tug2020. Readers who did not attend the conference should take a look. Two that I particularly enjoyed were by Amelia Hugill-Fontanel and Paulo Cereda. Peter Flynn's presentation on creating virtual bookshelves for use as background while Zooming was a lot of fun (they can be seen in the background of Jennifer's poster on the previous page). The conference proceedings, the present issue of *TUGboat* in which this report is included, will be entirely open access towards the end of this year. The proceedings includes papers or abstracts from all the presentations.

I was struck by the evolution of who is involved with what in the TEX world since I first attended a TUG meeting in 2003. Projects go on and evolve, while who is maintaining or developing them slowly changes. It is clear that TEX et al. remain in widespread use, there is a vibrant community, and that the end of TEX is not imminent.

Reflections

There were over 250 participants; in-person TUG conferences rarely reach 100 participants, more commonly 50–60. Being free and not requiring travel for participation are two evident reasons for the greater attendance. Twice there were 130 people in a presentation, split roughly two-thirds/one-third between Zoom and YouTube. There were seldom less than 40 people in a session regardless of the time of day. (Many more people registered than attended; presumably some will watch at a later time.)

The conference ran impressively even through occasional little glitches. Over the course of the three days, the hosts knew better what glitches to anticipate and to try to avoid. Non-host panelists could also unmute briefly to provide helpful ideas of getting around glitches. There was one unpleasant Zoom bomb — hard to avoid with free registration.

Recording presentations in advance was probably a first experience for most participants. They seemed up to the task. In addition to decreasing the chances for Internet problems, prerecording tended to result in graphically nicer presentations which may have been better organized than without prerecording. Also, very usefully, they didn't overrun their time slots as live presentations can do.

Having the conference be virtual had other advantages. It was nice to be able to leave the "meeting room" without anyone knowing. It was nice to do other things while in the meeting without appearing rude to the speaker, for instance, to eat, do other computer work, or tune into a baseball game on a separate screen.

There was lots of side chat during presentations, mostly extending from something a speaker had said. This did not seem disrespectful to the speaker as it would have in a live meeting room. The chat was often useful, such as someone giving the url of a great example of something the speaker had mentioned.

Holding this online conference on short notice required massive volunteer dedication and effort, which I am sure was greatly appreciated by the TUG board and conference participants.

Futures

There is hope that the 2021 TUG conference can be held at the Cary Graphic Arts Collection at RIT; of course it will depend on the global health situation. If not, TUG has learned a lot about how to have a successful online conference and the next one can be even better. If it can be live at the Cary, some observations from this year, such as the benefit of recording presentations in advance, could still be applied; there would likely be provision for some remote presentations.

The world is moving increasingly to digital communications. Recovering from having to cancel this year's in-person conference gave TUG a start to where the world is moving.

◇ David Walden
walden-family.com/texland

TUG 2020: A report and future recommendations

Paulo Ney de Souza

This document expresses my own opinions, and lacking, due to time constraints, are opinions of viewers and attendees of the conference. These are my recommendations for future online meetings and even in-person meetings with an online component.

The organization

The entire conference committee, Karl Berry, Jennifer Claudio, Rohit Goswami, Robin Laakso, Ross Moore, Will Robertson and Boris Veytsman worked very closely and diligently due to the time constraints, the cancellation of other TEX meetings and the wish to deliver a good quality program, despite the uncertainties of the COVID-19 pandemic around us. Two other TUG board members, Norbert Preining and Arthur Rosendahl, joined us for diligent work and late night meetings that made this all possible. I am forever indebted to them all and several other volunteers.

Early on we were grateful to attract an innovative and strong program, creating a series of conversations online and two keynote addresses with a broader appeal. Steven Matteson from Monotype and John MacFarlane from the University of California, Berkeley joined us with two master addresses. Javier Bezos and Philip Kime participated with intimate conversations on what it is like to do work on TEX and how it really gets done.

A field of very strong lecturers followed and I am very thankful to all of them for making such a nice meeting.

Hard work is required, especially if we are not able to pay hundreds of thousands of dollars for meeting organization, as some societies are doing right now. As might be expected, there are many areas that we need to improve, especially "reaching out" to a public that have not been able to attend a TEX conference before.

This report is a humble attempt to try to start that conversation.

The workshop

We started out by organizing the traditional workshop in Beginning LATEX as TUG has done for many years. About 40% of the enrollees in the conference said they planned to participate; it was attended by 60 people at its peak. Cheryl Ponchin and Sue DeMeritt have recorded many hours of video lectures showing basic techniques and usage in both TEXmaker and Overleaf. We played three hours of it as part of the workshop.

Thirty out of the total stayed for the whole three hours, with a clear over-representation of Latin America among the attendees — most likely due to the timezone we used for this part of the conference. Attendees were engaged and the chat was used for discussion not only of the lecture, but of the methods and tools used by Cheryl and Sue.

A few of the students have commented that they watched and engaged the workshop with closed captions and automatic translation on YouTube.

My recommendations would be:

¶1. Include an Intermediate Course, expanding on the collection of videos we have started to build.

¶2. Replay the lectures in intervals of 6-hours and 12-hours later to cover most of the globe. The challenge here is to find able bodies to answer the chat, especially outside Cheryl–Sue timezones.

¶3. Create a special parallel session on Zulip, possibly named "Ask the Expert" where people would for a certain number of hours be able to engage and talk to experts on specific topics of interest, for example, BIBLATEX, Ti*k*Z, ...

¶4. Create a library of instructional videos on LATEX.

¶5. Create a video about help resources for LATEX: `tex.stackexchange.com`, `learnlatex.org`, TopTEX, ...

¶6. Develop a library of instruction on specific topics, for instance, typesetting CJK languages in TEX.

The conference had a non-trivial share of talks on "learning LATEX". TUG could promote a special interest group in *Learning LATEX* to analyze these and possibly expand the reach of the Workshop. Needless to say, even though it is not on the mind of most TEXnicians, the beginner's workshop is an extremely important component of disseminating TEX. With that comes one further recommendation:

¶7. Build a SIG on *Learning LATEX*.

The conference

The program was strong and the eventual schedule of (more or less) 3 timezones in 3 days pleased all the speakers and especially the European audience. The presentation schedule, worked out mostly by Karl Berry, not only placed talks together by subject, but also offered them to speakers at a reasonable time.

The manual for chairs written by Will Robertson was fundamental to the smooth operation of the conference.

I was happy to be able to offer a document for speakers on creating and recording online presentations. With Ross Moore's processing into an accessible PDF, it remains available at `tug.org/tug2020/pres.pdf`.

Attendance

A total of 360 people enrolled for the conference and attendance at a peak was about 130, a mix of many old timers and some new faces, and quite a high percentage of anonymous viewers. (So the benefits of requiring a validated registration, mentioned below, need to be balanced against the reduced participation if anonymous viewing is not allowed.)

Coordinating a social scene complementary to the conference proved to be a bit harder than we expected. Most platforms imposed unnecessary restriction on enrollment, assignment and use. We moved from Zoom to Google Meet to GoToMeeting and finally settled on Zulip and Gather Town.

The entire social scene was set up by Rohit Goswami. We did not have much time to publicize it properly and redirect people there, but despite that, the Zulip instance had 80 people on it, some spirited discussion and hopefully people will continue to discuss there.

We ran a single ad for Gather Town at the top of the schedule page and it was well used given the timing. Groups of up to 15 people hanging out while some stragglers roamed free in the map (peak usage of around 23 people).

¶8. Evaluate the social interaction platforms to find something to complement the main platform used by the conference.

Technicalities

Next I move over to some technical issues that we should address in the interest of consistency, decreasing the workload and put more reliability in the process, by introducing more automation. This section and the next three are intrinsically related.

Zoom decides on the resolution at which to record a meeting depending on several factors including the resolution of the original source material, the server that is playing the material, and even available bandwidth. Because of this and some other limitations — one being the impossibility of muting yourself while playing any material — Zoom should be run on a server by itself, and not mixed with other tasks of the chair such as tending to email, chat or keeping the schedule of the conference.

We ran the whole conference on a flatfile database, but the frequent need to deal with:

- different roles for the same person
- talks being given by more than one person
- frequent calculations with time
- connection with Zoom API for controlling access

points to the need for an RDBMS that would simplify and automate various tasks. It would also systematize the development of support tools by different contributors and be more maintainable in the end.

We ended up having to deal with time in 4 different timezones: PDT, UTC, CEST and the attendee's own timezone. Most of this would have been easier with an RDBMS.

This time around, integration with Zoom and `researchseminars.org` required intense copy/paste of data, treatment of spreadsheets, etc., all of which would be simplified. So proceeding with the list:

¶9. Run the Zoom server on a dedicated machine — not laptops.

¶10. An RDBMS to make some of the services easier to implement and maintain.

¶11. Tighter integration with Zoom.

¶12. Better integration with `researchseminars.org`.

Given the short time to prepare for the conference, we did not have time to evaluate the alternatives to Zoom and Google Meet, especially open source platforms like Jitsi. Since Zoom was made available in partnership with the University of Adelaide, it is quite possible that other options are not going to be a match for that, due to the necessity to rent, set up, and maintain a good server. Nonetheless, since we are strong open source supporters, it behooves us to:

¶13. Evaluate the use of Jitsi (`jitsi.org`), MIT Unhangout (`unhangout.media.mit.edu`), and Apache OpenMeetings (`openmeetings.apache.org`) vs. Zoom.

Automation

We did automate many tasks, but need to go much further. Examples are the prompter for the talks, the schedule page and the generation of the title-cards.

The work that is needed *live* at the conference, such as beginning/ending a talk and running of the announcements for the upcoming lecture, required manual labor and login privileges by the chair, and so took noticeable time and was prone to error. In addition, the list of participants, issuing of certificates, and arrangements and classifications of paper, slides and videos was done by hand. They can all be automated with `cron` and a submission suite, and the chairs should only need to act in case the schedule gets late or some other altering occurrence, ideally via a web interface.

We should expand the prompter display interface to be a full dashboard showing all the aspects of what is happening automated in the background and how to revert anything, including small tasks like *record* and *stop recording*.

The line of thought here is to free the chair and hosts for the work of chairing and hosting. Recommendations are:

¶14. Automate the live talk on/off.

¶15. Automate displaying of the title-cards.

¶16. Automate list of participants.

¶17. Automate issuing of certificates.

¶18. Our server should automatically receive submissions from speakers and classify slides, preprints and movies for the talks, put them in the right places and update the schedule accordingly.

Redundancy

All chairs had access to all files necessary via Dropbox and this worked well, but last-minute impossibilities and even an unsuitable network connection can make the work of a chair very painful. To accommodate for that we should train and should always have available a replacement chair and a co-host:

¶19. We should have redundancy of chairs for every session — disasters can happen!

The website

The website now has a huge lack of conformity. Displaying the times in the reader timezone did require a bit more information on the page and a framework. This discrepancy should be resolved and some conformity brought to the display of this new information.

¶20. Make all pieces of the website consistent.

Social media strategy

We did not have much time to build a social media strategy, either for advertising the conference or to promote the meeting among possible attendees, and a last-minute emergency with one of the organizers almost spelled disaster. We were rescued by Rohit Goswami, who quickly built on the efforts by Jennifer Claudio and added the social rooms to the meeting. These should all be tied to the usernames used to enroll for the conference.

¶21. A social media strategy for publicizing the conference in at least four channels:

(a) TEX user groups worldwide
(b) Our own TUG membership
(c) Facebook
(d) Twitter

¶22. Automated Twitter and Facebook feed for every talk.

¶23. Ads on StackExchange.

I also emailed every author of articles in *TUGboat* over the last two years and that was an arduous task. Having a submission sequence would make that an easy operation.

¶24. Invite submissions by authors of *TUGboat* and other TEX publications.

More volunteer help

More volunteers with the conference are always welcome. We can use help managing the chat streams on Zoom, YouTube and Zulip during the workshop and conference:

¶25. Manage the chat in YouTube and feed back into the Q&A.

¶26. Help with the chat on Workshop.

¶27. Help with "Ask an Expert".

Work coordination

Assignment and cooperation of work among volunteers is always a hard issue because of people's availability and set of skills, nonetheless we should use systems that permit a more transparent and easier to deal with list of assignments and expectations.

The back room chat of organizers in WhatsApp was fundamental to solve a few problems. Same goes for sharing all files for the conference in Dropbox.

¶28. Use a task management like Asana, Trello, etc., for work coordination.

¶29. Use preset online meetings for touching base on difficult issues.

¶30. Texting on WhatsApp for backroom of the conference organizers.

Speakers

The work and interaction with speakers is not simple. We have spent more time processing 5 (problematic) talks than the other 35 added together. Some stricter guidelines are called for because of the added work necessary to check the videos.

The live talks worked wonderfully well, but to keep things consistent in Zoom, we should give stricter directions to speakers on setting the resolution of their monitors. That will help obtain a smoother and consistent recording of all talks.

Some talks are written to be pre-recorded (Paulo Cereda's were two excellent examples of that) and some are best live (for example, Ross Moore's talks). Thus the choice should be left to the speaker, but

with the request for a pre-recording whenever possible to cover for a no-show or technical problems at the time of the talk.

We should also invest in identifying pre-recorded talks that can be used to fill-in for something unexpected in the schedule.

The speaker–prompter built by Norbert Preining worked well in reminding all speakers of their slots as their time approached, and should be expanded, as noted above.

¶31. Move up deposit date for video recordings.

¶32. Strongly suggest prerecording, even if for backup purposes.

¶33. Stricter technical guidelines for online talks sharing video.

¶34. Meet with the speaker over Zoom on the setup that will be used for the talk.

Preservation

At present, the original videos are archived by Norbert Preining (who did much of the processing work on them). This is fine, but we should probably make it official:

¶35. Define an explicit strategy to archive videos.

Video processing

The processing of the videos should start immediately after the recording is made available by Zoom so we can shorten the time to get them up on YouTube. And some of that could involve the speaker, namely:

¶36. Allow speakers to edit their own closed captions.

Live discussions

The interviews seems to have worked, despite the lack of a good interviewer. Perhaps we could do the interviews in advance, and find a knowledgeable transcriber/editor to help make them available as text at the conference. We should make that a staple of all meetings, and even go further, with some panel discussions, as has happened at past TUG meetings.

¶37. Continue with the interview sessions.

¶38. Organize round table discussions.

◇ Paulo Ney de Souza
paulo (at) berkeley dot edu

TEX conferences and General Meetings, this year and next

Jonathan Fine

I'm writing about our TEX Conference next year (2021), and our TEX Conference and General Meeting this year (2020). A satisfactory result in difficult circumstances counts as a success, or perhaps even better. So adverse were the conditions for this year's TEX Conference that it must be counted as a success, even though flawed in several important ways.

A major flaw is that ordinary participants had little opportunity to speak after the talks, or with each other at other times. In short, for most participants the social aspects were limited. In addition, there was not a TUG General Meeting. So far as I know, every previous TUG conference had a General Meeting as part of the event.

On 5 May 2020 the TUG Board decided unanimously to cancel both the TEX conference and the TUG General Meeting (both scheduled to take place in-person at RIT, Rochester, New York). On 2 June we were told that the TEX conference would go ahead online. (There's no recorded Board decision enabling this.)

It would be unfortunate if, in 2020, TUG does not have a General Meeting. Such a meeting is, I believe, a formal legal requirement arising from our bylaws and also our non-profit status. However, it's not too late. We can hold the General Meeting later this calendar year!

Many were surprised that our community was able to hold an online conference, for the first time, and in these adverse circumstances. I'm so pleased that so many of us, including myself, went outside our comfort zone, and contributed what we could. This includes a wider than usual range of speakers.

And together, this was enough. Once it became clear we would overcome adversity, and the conference would happen, then doubt and fear reduced, and we became stronger and more capable.

Much of the credit must go to Paulo Ney de Souza, particularly his calm and supportive presence, his commitment and initiative. It seems to me he was the main force and lead for the organisation of the conference this year. He did the same for the 2018 TEX conference that coincided with the International Congress of Mathematicians in Rio de Janeiro. Paulo has never been on the TUG Board. I think this energy and view from outside was helpful.

Whatever the public health situation, having a large on-line component to the 2021 TEX Conference would be very valuable. It will increase attendance

and diversity. It will enlarge the community. And where allowed, as part of the Conference we can meet in small socially distanced events. This would help restore the social element to all of us. I'd like an international on-line General Meeting and social events to be part of next year's Conference.

I can think of nothing more important now, for ensuring success next year for both Conference and General Meeting, than there being an online General Meeting this year, to follow on from the Conference. Let our present success be the foundation for better circumstances and outcomes next year!

◇ Jonathan Fine
jfine2358 (at) gmail dot com
jfine2358.github.io

Interview with Javier Bezos

Paulo Ney de Souza

Editor's note: This interview took place on 25 July 2020, during the TUG 2020 online conference.

Paulo Ney de Souza: Nice to meet you, Javier. Your first time, and half in person.

Javier Bezos: You're welcome.

PN: First let me introduce Javier to the people here that may not know him. 10 years ago he has assumed maintenance of one of the most basic and most famous packages of LaTeX, which is Babel, and is also someone deeply interested in issues of production and use of TeX in production, and so forth.

Welcome to TUG 2020. Let me see if I can start shooting you some questions.

First thing I'd like to hear is your background. How did you grow up, how did you get here? Can you tell us a little bit about your background, independent of TeX?

JB: Well, I was born in the Canary Islands, although I spent most of my life in Madrid. I think there is nothing special to tell, or it doesn't seem so to me.

My liking for language issues and the world of books, go back to my high school days because I was one of the founders of the school paper, and I was passionate about having a nice look with all the letters in their places, are the types right, ...

Then my parents asked me if I'd like a Spectrum.[1] And this changed my life in many ways, because I discovered something for which I had a natural intuition. I wasn't interested in games. They quickly bore me, and I began to program. I programmed a little word processor, and then I started to write in assembly language. When I remember now how much my parents spent in books on the Spectrum—; they were difficult to find, and very expensive. I'm very grateful to them for supporting me in almost anything when I wanted to dedicate myself to something.

When I had to decide what to study, the first thing I thought was physics, physical science. But I suddenly changed my mind. And finally, although it seems a bit strange, I decided to study radio, and that's what I did for seven years in a classical music radio station.

But even then, I didn't stop my TeX jobs and other subjects and actually right now my main activity is as linguistic advisor, which has little to do with the radio, but I am a person with very broad tastes.

PN: Can you tell us how did you meet TeX for the first time?

JB: Thanks to my wife's thesis. Wife now; she was my girlfriend then. Before, I already worked with Ventura, an incredible program, and I must confess that at first, LaTeX seemed incomprehensible to me.

I got the Knuth book and it looked to me like a mess. But I could cope with the book by Kopka and Daly, and there I began to understand what it's all about.

In Spain, there was no TeX group, so I signed up for DANTE. So in addition to learning more about TeX, I practiced a little German.

I already said I like programming. So I started writing small macros, then not *so* small macros, and finally packages. I usually say that there are those who solve Sudoku to entertain themselves, and I program. Especially, but not exclusively, with TeX.

PN: Interesting, interesting. What changes have you seen in the document production world and in the TeX world during your career, since this time that you began, all the way to now?

[1] An early UK home computer from Sinclair.

JB: The love for a well-made book at all levels, doesn't seem as popular as it was years ago. The phrase "Go forth now and create masterpieces of digital typography" doesn't make a big impression today. I also notice there is a great tendency to believe that typographic skills are built on the handling of a program when that's just the starting point. The point is, design is regarded more as the creative aspect than the practical one for the needs of actual documents.

But let's be fair. We can't say the love for a well-made book was universal formerly, either.

PN: Aha. I had an interesting discussion with Boris yesterday, and it seems that TEX, what with the size of the user base, TEX is one of the world's oldest subsystems. And that is 1977 to now, quite a few years. Do you think that it will continue forever?

JB: I think now there may be a little revolution going on with the arrival of LuaTEX. It has its loose ends, but it's an important advance, and its possibilities remain to be seen.

In fact, I'm using it a lot in Babel, because it has allowed me to introduce functions for non-standard hyphenation and for transliterations. Not to mention bidi writing and line breaking where we can have almost absolute control.

PN: What packages, what kind of tools do you find essential to your work right now as a document author, as well as a producer; what do you do use on a daily basis?

JB: I don't think that there's a simple answer for this. When I start reading LATEX package manuals to see what they are all about, I'm still amazed at what they can do. And sometimes I wonder how I've been able to live without a certain package.

The packages I use the most are the standard ones: `hyperref`, `graphicx`, my own packages, and also a package by Zdeněk Wagner for the page layout (`zwpagelayout`).

PN: And so when you're producing a book do you use exactly the same thing as an author?

JB: No, it depends on the project...

PN: And what do you use mostly with that flow, to produce your own work? Do you use LuaTEX mostly for producing your own work?

JB: I usually write my own macros. So I like to reinvent the wheel.

PN: [laughter] Let me ask you this. You are a guy that knows a lot about other issues, especially things like typography and orthotypography. Do you think it's worth spending a lot of time on certain issues in typography, even though very few people notice?

JB: Oh, sure, they notice it, although without realizing it. I want to look at books which invite reading.

The problem is there are publishers who believe the opposite, and not only in the field of typesetting, but also, for example, in copyediting. I don't know how the situation is in other countries, but in Spain, many media have dispensed with copyediting.

PN: Aha, so the production is substandard, as we evolve into the 21st century.

JB: Yeah.

PN: Do you use other tools besides TEX in your work?

JB: In my day job, no, technically no. As I'm mainly a linguistic advisor, I use it for complementary tasks. I work at a foundation called Fundéu, directly linked to the Efe news agency, and partly to the Spanish Royal Academy, whose function is to help the media in their use of Spanish.

But there can be a connection, because writing and pronunciation of names from other languages is another of the tasks of the foundation, and I'm something like the specialist in romanizations. So I can see there is a connection, even if it's more between my daily work and what I'm doing in Babel.

PN: You do a lot of work on a certain area where not very many people understand exactly what it is, which is orthotypography. How did that come about, how did you become interested in such a specific area of typography? But first, would you explain to us what orthotypography is?

JB: Basically, it is not very different from a combination of orthography, style and typography. I'm not sure exactly what orthotypography is, but well, in Spain it is a concept you should use very often. And if I'm right, in French too.

PN: On this point of orthotypography, I want to bring an issue which is very dear to mathematicians.

There are two camps in mathematics in terms of punctuation and in formulas which are central to a page. There's the people that believe that it *should* be punctuated, and there's the people that believe it should *not*, that punctuation is to use a pause, and that, and that if you have a formula which is centered on the page, that pause already exists.

What is your idea on that?

JB: If you put it when the formula is in line, at the end of a sentence, why shouldn't we put it when displayed?

Anyway, in typography, what matters is to have good taste. Good taste and good eye. Then we can have your rules of style. But if they don't work for us in a certain context, you shouldn't hesitate to think of other possibilities. After all, it's still

an art, like painting, like composing music. Music may follow many rules when we are learning, but the masterworks are being created by breaking the rules.

PN: Interesting. Very interesting.

Let's talk about Babel a little bit. I mean about 10 years ago, a little bit more than 10 years I think, you took over as the main lead maintainer of one of the most used packages. How has it been? Tell us.

JB: My interest in the localization issues came from much earlier. Already in 1997 I was considering creating an alternative to Babel, which I called Polyglot, but it wasn't long before I realized it was a futile effort because Babel, after all, mostly did the job, and it was the standard.

Then came another opportunity, Omega, by Yannis Haralambous and John Plaice, and I started working on it. This time, the package was named Mem, a Hebrew letter, but we know that Omega sadly didn't develop its full potential. So Mem also failed.

Then Frank Mittelbach proposed to me to join the LaTeX team (I think that was in Brest, EuroTeX 2003), which I gladly accepted because that way I could help in this area, and possibly others as well. And I still continue.

PN: How do you do your work on Babel? How do you connect to authors, languages, and the production of the `.ldf` files ... could you tell us a little bit more about that workflow? How do you find the people to help out with the languages?

JB: In many cases, I generally don't like interfering with the work of those who develop national styles, some of which are supported by their respective user groups. But I'm always willing to help and collaborate. Also, if I discover there is someone interested in supporting some language, I contact him, as I did for example with the group for Kurdish.

On the other hand, I like to investigate by myself. I'm self-taught, self-taught by nature. And I like to learn things by myself. I think I'm good at it and I really enjoy doing it.

Sometimes it means you end up reinventing the wheel. Sometimes that wheel is worse than the others. Other times it turns out to be better. Of course, that doesn't mean I don't like suggestions, help or whatever. Quite the opposite.

PN: I was doing some research a few weeks ago on Babel and other things that you have written in order to understand your work a little bit more, and I came across with a package that you maintain for the language of Guaraní. This is my mother's tongue. Literally. My mother was a speaker...

I was extremely surprised to see this. This language is spoken by a lot of people, but without formal production of books. It's a language spoken in South America, from the north of the Amazon, all the way to the south, by indigenous people of South America.

How did you connect with this language?

JB: So many years ago. I just don't remember why I did it. It's true that for a time, I was interested in the languages of the Spanish-speaking countries, but I don't remember more details, unfortunately.

PN: Ok. I guess one question that a lot of people are interested in is about HarfBuzz, how LuaHarfBuzz is changing, and may change Babel.

JB: Very little, with relation to the original LuaTeX which I have been working on for some years.

Mainly because HarfBuzz is about fonts, not about languages, and sure, the fonts contain the information about how to render a script, but I think HarfBuzz is not exactly part of Babel. More of the LaTeX kernel.

There is a possibility to make HarfBuzz the default shaper, but as Marcel [Krüger] explained, I think it's not a good idea. Because there is a problem with how HarfBuzz deals with hyphenation points in the Latin, Cyrillic and Greek scripts, because many hyphenation points are lost. It depends on the font, but in those scripts in most cases the best option is to stick to the default renderer.

PN: Where do you see the future of Babel?

JB: Serving as a framework to localize and internationalize documents at several levels. Until now, Babel has followed a model which I call *vertical*, based on language styles which are basically self-contained black boxes. That's fine in monolingual documents, and in fact is and will continue being the recommended method in these cases.

But when we need to combine several languages, we can have serious internal relationship problems. So I'm working on a new model, which I call *horizontal*, where languages can be defined in a descriptive way, with the help of `.ini` files based on key–value pairs.

That allows us to reuse a solution for one script in another without too much hassle. For example, traditional line breaking in Amharic turns out to be almost identical to that of a southeast Asian script. So it was enough to adjust the keys to get the right result.

PN: I have a specific question that comes from the mathematicians' camp... will there be a day in which we will be able to choose mathematical functions based on their own languages?

JB: It's something that has been requested sometimes, but on the other hand, it's a controversial issue — should they really be translated? And after

all, it can be solved easily with `\renewcommand`, so I wonder if it's something necessary in Babel. Well, I think it should not be ruled out beforehand, but it's not one of my priorities.

PN: The biggest complaint I hear, Javier, is that if you have a big shoe, your foot grows, and then you start seeing books in Portuguese with "sine" instead of "seno".

JB: The same in Spanish, yes. And there are various others written with an acute accent, traditionally.

PN: So let me ask you about some modern questions. One of them is about Polyglossia. Polyglossia was sort of started right around the time that you became a development leader on Babel, and Polyglossia got started as a replacement for Babel for XeTeX and LuaTeX. Then the development of Babel picked back up again with your work. Do you see some synergy happening between the two packages?

JB: Well, actually Polyglossia is older, about five years I think, and on the XeTeX list some people even suggested I take charge of it, but I did not for several reasons. One of them was my personal situation at the moment, but another was my experience with Polyglot, which I mentioned earlier.

By the way, if you wonder if these names are related, the answer is yes, they are. There was discussion about a multilingual package for XeTeX and I proposed this name because I wasn't going to use it any more. And that name, but in Greek instead of English, stayed. I don't remember who proposed the translated name, but it was a good idea.

When I created all the new `.ini` files, I did it thinking they could be useful for both Babel and Polyglossia as a shared resource, and not only for them, but also for other packages.

Well, this work was done and now we need to fill it, which I'm doing gradually. There are already about 200 more-or-less completed languages or easily completable, and there are also templates for about 500 languages, and they can be a starting point.

PN: So I see that you are, you seem to be fully BCP 47 compliant, at least on the naming of the files and so forth.

JB: Yes, sure, we can already deal with BCP 47. What's more, you don't even need to declare languages in the preamble, because Babel can load the locales on the fly, with some limitations.

Thinking about use cases, I realized there are texts generated externally, and the main document may not have the slightest clue of the languages needed. An example can be a bibliography, with many names and titles from a multitude of languages.

PN: Let me ask you a question now about preproduction and production using PDF. The open source community tools seem to be completely unprepared for dealing with prepress. We have ways to check PDFs, we have problems with resolution, colorspace, flattening, transparency, overpainting, and I see that you started doing some embryonic work on the package `colorspace`. Can you tell us where were you going then?

JB: Yes, this is a problem. I'm sure the work in progress will bear fruit. For example, Joseph Wright and Ulrike Fischer are working on improving the color support in LaTeX, so that what `colorspace` does will be better done and avoiding the conflicts between different packages trying to generate PDF code directly.

On the other hand, we should never think a document created with, say, InDesign is automatically ready for printing. I myself have suffered from poorly created documents by designers who think it's all about creating frames and things like that. They may not even know how to correctly import a PDF image.

Here again, I think that LuaTeX could be of great help. For example, I think it should be possible to write a utility to convert calibrated colorspaces. I'm not sure, but I think ConTeXt can do it.

PN: That's great news. LuaTeX can be used for reproduction. So do you plan to take `colorspace` into a full color production package for PDF?

JB: For the moment, my work on `colorspace` has stopped. There is no point in continuing with it if its features will be supported directly in the LaTeX kernel.

PN: Thank you. We have a question from Michael Topp, on the net. What compilation engine do you generally use or recommend?

JB: I use LuaTeX since many years ago.

PN: Thank you very much, Javier. I really appreciate your coming for this interview with us.

JB: Thank you.

◇ Paulo Ney de Souza
`tug.org/tug2020`

A Addendum: Recent advances in `babel`

Javier Bezos

The purpose of this addendum is to highlight some of the progresses made in `babel` for it to become, as explained in the interview, a localization framework. New features are announced and explained in some detail in the `babel` wiki, `github.com/latex3/babel/wiki`.

No program can be considered finished. The world continues to evolve, and there will be areas deserving improvements. So, expect new features in the future. For example, at the time of this writing, there is work in progress to deal in a coordinate way with chapter counters (Hungarian, CJK), lists and other labels.

Locale data. There is a large database with `.ini` files directly usable by `babel`, based in data taken from several sources, including the Unicode Common Language Data Repository.

I think these files will help to solve what I think is one of the most common problems when creating a language style based on TEX code, namely, there are linguistic communities without a TEXnician at hand to develop it. There are templates for about 500 languages in the GitHub repository, which can be used as a starting point.

One of the main advantages of the `.ini` files is that the data is provided in a descriptive way. For example, the file for German (`babel-de.ini`), contains things like:

```
[identification]
...
name.local = Deutsch
name.english = German
name.babel = german
name.polyglossia = german
tag.bcp47 = de
language.tag.bcp47 = de
tag.bcp47.likely = de-Latn-DE
tag.opentype = DEU
script.name = Latin
script.tag.bcp47 = Latn
script.tag.opentype = latn
...
[captions]
preface = Vorwort
ref = Literatur
abstract = Zusammenfassung
bib = Literaturverzeichnis
...
[date.gregorian]
date.long = [d].[ ][MMMM] [y]
date.short = [dd].[MM].[yy]
months.wide.1 = Januar
months.wide.2 = Februar
...
```

I invite all linguistic communities to complete them, and to make suggestions and feature requests for any language to improve the LATEX support in the three main engines. Feel free to fork the GitHub repository and to make pull requests.

BCP 47. BCP 47 tags can be optionally used to select languages, but note since a two- or three-letter word can be a legitimate language name, BCP 47 codes are not activated by default. In fact, the BCP 47 tags have been defined in `babel` for years, but they were not easily accessible nor usable. In the most recent versions, you can select a language (its most basic features) merely by writing something like `\foreignlanguage{zh-Hans}{...}`.

I wouldn't like to go too fast in a more or less complete implementation of the subtleties of BCP 47 tags. I prefer to go step by step, much in the spirit of continuous development, but now the most basic tags are already there.

Line breaking and bidi writing. Thanks to LuaTEX, `babel` supports bidi writing without explicit markup and the three basic methods for line breaking: CJK, southeast Asian and Western. You can even change the properties related to them for specific characters; for example, the direction may be changed from 'Arabic letter' to 'Other neutral'.

Automatic font and language selection. Sometimes, you need to insert short pieces of text in a different script from the main one; for example, some Bangla words inside Bulgarian. Now, language identifiers (`\language` and `\localeid`, which currently apply mainly to line breaking) and fonts can be automatically set based on script blocks, without explicit markup.

Transformations. Another feature related to LuaTEX, still under development, although the basic features are already usable, are the commands `\babelprehyphenation` & `\babelposthyphenation`, which allow transliterations and non-standard hyphenation. For example, with the following settings "ff" is automatically divided as "ff-f", and also "mm", "tt", "rr", "pp":

```
\babelposthyphenation
  {german}{([fmtrp]) | {1}}
  {
    { no = {1}, pre = {1}{1}- },
    remove,
    {}
  }
```

Counters. They can be defined in `ini` files by enumerating the elements:

```
lower = a b c d e f g h i l m n o p q r s
```

The same applies to additive numerals; examples are Greek, Syriac and Japanese.

◇ Javier Bezos
jbezosl (at) gmail dot com

Interview with Philip Kime

Paulo Ney de Souza

Editor's note: This interview took place on 25 July 2020, during the TUG 2020 online conference.

Paulo Ney de Souza: Nice to meet you.

Philip Kime: Yes, a pleasure. It's nice to put faces to all the names I see on the StackExchange.

PN: Exactly, exactly. Can I go for the first question?

PK: Absolutely, please do.

PN: Well, I'd like to start out with some of the stuff that I really learned from David Walden, which is to ask people first to tell us a little bit about your background.

PK: Well, it is a fairly straight sort of academic background in Philosophy. I graduated in Philosophy in, gosh, 1988, the University of Warwick in the UK. Then I went on to teach in mostly Analytic Philosophy and Logic, Metaphysics, Philosophy of Science, that sort of thing, and then I went on to do Artificial Intelligence, in Edinburgh, a Master's degree, and then I went on to do a Cognitive Science PhD in Edinburgh. After that, I sort of had the usual pile of student debt. And so I went on to do IT consultancy to pay off the debts.

This was in the boom period, mid, late 90s in the UK where, if you had a keyboard, you could become an IT consultant. So I did that, and I sort of bluffed my way into a consultancy job with British Telecom, by literally reading a Nutshell book on TCP on the bus on the way to the interview. Luckily they asked me questions to do with what I had just read in the book on the way there. I got the job but I don't remember well what I was doing. We were hacking some Unix systems for British Telecom, and this was 1998.

So I stayed in that sort of area for several years to pay off debts and moved around and did a few years at HP as an external consultant — not as an HP consultant, but as an external contractor working on an HP site in Scotland. And I knew I didn't particularly want to be doing that forever. It was just useful to pay off bills and then I, I mean my background is Philosophy and I'd been reading a lot of Philosophy, a lot of Psychology, and I ended up looking into becoming a Psychoanalyst. So at that point I went through the stages of looking at where to go to do this. After a lot of reading, I wanted to follow the general Jungian approach.

It turned out the best place to do this if you wanted to still work and actually earn a living was to go to Switzerland to the original Jung Institute. So that's what I eventually planned to do. So I started to do the preliminary work for this. And then there was an IT crash in the UK, early 2000s. And so basically I just ran out of money and then I just had to give up the idea of going to Switzerland and I went to work in the Netherlands and then in Belgium for a while, saving up money and then I eventually said, well, it's now or never. And then I moved to Switzerland and started the training at the Jung Institute which went on for quite a few years with me working in IT, and going back there in rotation. I eventually managed to find a job in Switzerland in IT, so I could do both at the same time. That went on for a few years, and I got married and moved to the US for a few years and did clinical work there and then moved back to Switzerland and a lot of back and forth and messing around. I settled back in Switzerland in 2008. I still did some consultancy IT work, but I finished off the training there at the Institute and eventually I was on the board of directors of the Institute and then became the Vice President for a while.

Now, as a second career, I have a private practice as a Jungian Psychoanalyst in Zürich. So, yes, that's my somewhat peculiar background.

PN: Wow, that's quite a journey.

PK: A lot of moving around. I'm sick of moving countries. I hate it, I really hate it.

PN: How does TEX fit into it? When did you get interested in TEX?

PK: That was in '91 when I moved to do my Masters degree in the AI department in Edinburgh at the time ... It's funny to think of it because the department actually burned down years after I had left. It was a big story at the time. It was one of the oldest AI departments in the world, one of the first, and it burned to the ground. I think this was, this was quite a few years ago now. Everything, including the only copy of my Master's thesis, went

up in smoke. *Everything!* This was a big disaster at the University as I recall.

Because they were a very traditional sort of AI department, they had only HP Unix machines. There were no Windows machines, no Word, just old HP Unix machines, and everything, all documentation had to be done in TEX, in LATEX. There were no nice GUIs anywhere, only X Windows. It was all running in black and white; there were no color screens. It was a very hard core, old school AI department. Everything *had* to be done in LATEX. I mean it was just assumed, that was how we were trained to write all documentation, do your thesis, everything. So it was just sort of from day one, we were just pushed into that and I quite liked it because it was, you know, it was documents as code and I quite like that. And so, and this was in the early days before LATEX 2_ε, so that was my Master's degree. We were just forced to do it. We were forced to do it and it was, you know, relatively fun really, and it was the same when I did my PhD in the Centre for Cognitive Science at Edinburgh, that was the same. The only difference was they had Sun workstations instead of HP workstations and everything was LATEX, my PhD was in LATEX, everything. And so there was a very, very hardened LATEX community there with lots of hackers. And so it was an easy environment to get into it with.

But when I left that, then you know this was in the days before there were any reasonable home distributions of LATEX. There was no MikTEX really at the time and no MacTEX on Mac. Nobody had Macs at that time in the UK anyway, it was too expensive. So I went back to using Word, which was just horribly painful. I mean, I remember just hating it for years, and it wasn't until I started writing the thesis for the Jung Institute in about 2007, that I thought I'd investigate LATEX again. So those are significant years in the wilderness of not using LATEX.

When I went back to it, I found these marvelous distributions. I found MiKTEX. I moved to Macs because of the Unix subsystem. I found MacTEX and said, Oh! And I got back into Emacs again. And so I started using it, there was a whole phase where I moved away from it, and then I came back to it — redemption in about 2007.

PN: How do you mix the two things? You work in psychiatry and you work in TEX?

PK: I don't; I don't really. I mean, I still do IT consulting. So, but even there, you know, it's not used in most large companies anyway. So I don't. It's a totally separate thing and it's quite nice because I always considered the sort of IT side of things and sort of computing to be a nice concrete non-abstract compensation to Philosophy.

I still do quite a lot of work in Philosophy; Philosophy and Psychoanalytic work is very nonlinear, and you don't generally get nice clean solutions in anything. So it's *nice* to come to a computer where somebody says, here's a bug report and you can just pick something and it's fixed, unlike with people. And so this is, it's a nice compensation, but it has literally nothing to do with the other side of things. I mean, I do all my invoicing and things for my practice myself in TEX, but it's not part of work.

PN: Aha! Are you are you doing e-consultations during the pandemic? Are you...

PK: Oh, yes, yes. I mean, in Switzerland, they did it all quite well, they planned it quite well, they're very, very organized in Switzerland, so the health authority immediately got on top of this and authorized, you know, remote sessions immediately for everybody. So yes, I basically stayed at home a lot and did remote sessions with Skype with people, which was actually relatively straightforward.

PN: Can you take a consultation right now?

PK: For you? Yes, yes, what's up, what's wrong?

PN: Well, I mean, I feel extremely tired attending these lectures online.

PK: Oh, yes.

PN: Probably my fourth or fifth conference, and I kind of blame it on the lack of connection with, the lack of seeing the full body language of the person.

PK: It is absolutely, I mean there's research on this. Yes, it's clear that the problem is with, with engaging with the screen. It's that, I mean, for example, I mean, I'm looking outside from my office and now my peripheral vision...

Well, most of my vision of what's going on is peripheral vision. I mean, I can see the whole room — the windows, the trees outside, everything. Right. And when you're physically in a conference and you're physically watching someone, you're generally just watching the person, and you're listening to what's going on in the room. But if you start watching a screen, this isn't the case. There's ambient noise and peripheral noise and distractions, and this takes... So you're basically parallel processing a lot when you're looking at online things, which just takes a lot more energy and it's not a little more, it's quite a lot of energy. I mean, you know, multitasking and multi-focusing and maintaining a sort of peripheral awareness of what's going on semiconsciously, just takes up energy. And so, I mean, online

seminars and online courses, it's all very nice, but they actually... There's some quite good research that shows that they're actually more exhausting in, in terms of concentration in cognitive effort because there's just an awful lot more peripheral distraction.

PN: Talking about peripheral distraction and peripheral noise, I have seen some other interviews of yours, where there is a beautiful sound of a clock that interferes when...

PK: Yes, the church. That's the church opposite. This is a very Swiss thing.

I live in a small village and I live right opposite the church, and this is one of those churches that makes no concessions to the modern world whatsoever, so it rings every single quarter of an hour. It rings once on the quarter hour, twice on the half hour, three times on the three quarters, and then it rings the hour, and it does that 24 hours a day without exception. And not only that, the special times which coincide with what used to be farming signals — that is, a lot of rural Switzerland is farming, it's still very farming oriented — so for example, if it rings, it rings a two-minute ring at six o'clock in the morning, again at 23 minutes past seven. And these tiny Swiss churches have GPS in them so that — I'm not joking — they have GPS, so they ring *exactly* at 23 minutes after. I mean, you can set your watch by the movement. And they do that also at one o'clock and, sorry, 23 minutes past one — I don't know what the 23 minutes past thing is — five o'clock, eight o'clock and 11 o'clock, and so it's constant bells. And I really like it now. I mean actually people visiting just said it drives them mad.

But when it stops and they do maintenance on the clock, I think that there is something wrong with the world. I think something's missing. What's going on? And you don't know what it is for a few hours. Oh, the bells are not ringing. So it's, you get used to it. So I really like it, but some people, it drives them absolutely mad.

PN: I just don't want to, I just want to advise our listeners that if it rings, you know, whoo! Stay put!

PK: Well, what time is it now? No, you're all right. It's only going to ring the hours and you won't really hear it in this room, but if there's a, you know, if it was 23 minutes past you'd hear it for a while.

PN: That's absolutely beautiful, absolutely beautiful.

PK: Oh, it's lovely. Yes.

TEX, BIBLATEX, and Biber

PN: So moving over to TEX. You already said a little bit about, you know, how your relationship to TEX evolved, but I want to ask a more specific question, how production of documents evolved during your lifetime. To me the first time that you started producing documents, up to now, because I see that you do produce a lot of documents yourself and...

PK: It's strange, because I, in some respects, I feel, I mean, apart from sort of development work on Biber and BIBLATEX, I don't actually... I'm not much of a sort of TEX, you know, power user really. I mean, I have a set of templates. I use bits and pieces for publishing papers and things, but, you know, I feel a bit of embarrassed sometimes when I see some of the incredibly complicated questions coming in for BIBLATEX, and I think, I have no need for this feature at all. I will never, ever use this, nothing even close. But I'm happy to, you know, make it work for somebody else. But I will never need anything that complicated because my needs are quite basic, really.

And in the humanities publishing world in journals and books, I do everything in LATEX. But then I always end up having to use some dreadful piece of software from Adobe or something that converts it into a Word file from PDF, because nobody will accept, you know, they won't even accept PDF. A lot of humanities publishers, you have, you have to send them Word documents, which is just awful. My document production went from doing a lot of complicated LATEX stuff for my PhD, with all sorts of stuff with this special logic system which required this special... It was a very odd notation, called (what was it called?) Discourse Representation Theory and Situation Theory, which had all these boxes and stuff. And we did all of it in LATEX. And the lecturers would provide special macro packages and it was quite complicated. And now, I mean, you know, if I have to put something in bold, it's exciting. I don't really do particularly exciting typesetting any more. So I get my sort of fix and exciting typesetting from working on BIBLATEX and reading TEX.StackExchange.

PN: I wanted to ask you a question about... You seem to have come to Biber first and then...

PK: Now that was a bit of a strange story, because I... So in about 2000 and... So when I was writing my thesis for the Jung Institute I needed to ... oh, no, no, it wasn't that. I was sending a paper to a journal and they required American Psychological Association bibliography format. And so, you know, I went back into BibTEX for the first time in years,

and I was looking around. I quite quickly came across BibLaTeX and thought, this is better, I'll use this instead.

And this was in the days when a chap called Philipp Lehman, the original author of BibLaTeX was around but he mysteriously disappeared about ten years ago. This is why I took it over. He was heavily involved in it when I started and I looked for an APA BibLaTeX style that would work but there wasn't one. So I thought, well, okay, I'll write the style for that. I started to write the style for the APA and then I came across some things that needed to be done to implement the APA style which I just couldn't work out how to do. And I contacted Philipp Lehman, then we had a discussion, and he said, well, you can't really do this in BibLaTeX, it has to be done by the backend, which currently is BibTeX. He then said that there was another chap who was implementing a special backend for BibLaTeX called Biber, this chap, François Charette, who originally started Biber, the very early phases of it. And he said, why don't you contact him and see how that's going. So I contacted him. And it turned out that he was, he was writing this custom back end for BibLaTeX called Biber, and so I tried it out. I found a bug with it and I submitted a bug report and he said, why don't you help me develop this because we need this. BibTeX's too limited for what BibLaTeX needs. So I said yes, and this was in 2009, early 2009.

And so I started helping him develop Biber a bit. Then it really snowballed and we started developing that together heavily in 2009 and that went on for a year or so, with us collaborating with Philip Lehmann on BibLaTeX.

And then François Charette didn't have the time any more, and he just left it to me. So I, ... and eventually it got completely rewritten, so there isn't a line of the original code left in Biber any more. That was sort of late 2009, early 2010, I believe. And so, yes, I was involved in Biber first, and then for a few years, it was just Philipp Lehman on BibLaTeX, and me on Biber. We were just collaborating on timing releases of both of them for a while.

And then he disappeared. He just stopped answering emails and I tried to contact him, and I still don't know to this day what happened. He just disappeared. And because I knew more about the BibLaTeX code than most people, because I've been closely involved with him working on it, there was really no choice. And I sort of, you know, did the whole messing around with GitHub and SourceForge at the time to get ownership of the project, and took it over in about 2012, something like that.

I want to say one thing, quickly here, before I forget. I want to say a big thank you to Moritz Wemheuer. I think he's probably watching us. I saw his name on the participant list. In the last couple of years, he's come on board the BibLaTeX team and he's done fantastic work, particularly with localization and styles and things like this. So he's really taken on, I mean, a serious load of the BibLaTeX development in the last couple of years, and without that it wouldn't have been possible, because it was getting too much.

PN: Are the two entities married to each other, Biber and BibLaTeX today? Or are there other clients for Biber?

PK: There are. There are other clients because Biber has this... I mean, basically, they *are* married, I mean they, they developed in lockstep and they're... I mean, you know it's, 90% of it is, is a complete marriage of technology. So it's not really Biber as a standalone tool. It's possible because we implemented quite a few years ago now a Tool Mode in Biber which allows you to take in `.bib` files and spit out `.bib` files and do various things to them in the process. So you can mess around with your bibliography databases without actually typesetting anything, and this is known as Tool Mode in Biber and it's reasonably sophisticated now. You can do all sorts of things to read, format and mess around with the data semantically. So there *is* a user base outside of that, but it's not particularly large, I would say.

Unicode and publishing

PN: Let me move over to Unicode and ask you a few questions. What are the hardest parts of dealing with Unicode and implementing sorting and making both of them understand Unicode well?

PK: That was one of the hardest things. However, I can't claim much virtue in this respect, because I use a very nice Perl module, which is part of the standard Perl distribution — the Unicode::Collate module — which was written by a Japanese chap, and it's very, very good. And that follows the Unicode updates and standards quite closely so the actual implementation of the Unicode collation algorithm, I have not done that because that would be insane. I submitted some enhancements and bug reports over the years just to get some of the features we wanted into Biber, but I basically use a library that implements the Unicode collation algorithm.

The difficulty was implementing a reasonable multi-field sorting algorithm for bibliography data. You want to have it so that people can sort bibliographies on arbitrary bibliography data in the most

flexible way possible, which means you have to have a proper multi-field sorting algorithm. So you want to be able to sort, for example, by author, then by title, then by year, then by volume, then by whatever, all of those. You want to be able to do those descending and/or ascending on each of the fields. You want to be able to turn off case sensitivity in each of the fields independently. You want to be able to switch your sorting locale, which we need, of those fields independently, all of which you can do with it.

And the difficult part also is, for names, basically constructing the data. The data structure for sorting is quite hard. The Unicode collation algorithm is very nice, and allows you to just sort arbitrary strings, but you have to somehow construct a data structure and the strings in that data structure in order to sort them. And that was the difficult bit, in order to do this in a consistent way. That took a few iterations. It's now quite a complicated data structure with a lot of optimizations in it so that we can short circuit sorting. It's by far the most compute-intensive thing that Biber does.

If you profile it, most of the time is spent in the collation algorithms for any large bibliography, so it took a bit of a while to get that sorted out. It was one of those painful excursions into sorting algorithms and things like this. I'm not really, you know, a low-level algorithmic hacker at all. So it was somewhat painful to have to do that.

But it's reasonably nice now and all I really have to do now is follow updates in this, in the modules that implement the Unicode collation algorithm and release new versions of Biber with the updated modules. And I don't have to mess about too much with the actual sorting algorithms themself inside Biber because they're quite stable now. As of about four or five years ago, I settled on a particular way of doing it, which seems to be fairly stable.

PN: And any updates, you just import through the Perl package maintained by this Japanese guy?

PK: Yes, yes. I mean, because all that really does is it just pulls in the latest UCA updates and the latest key generation algorithm in the key generation data files and so I don't really maintain any of that. That would be, that would be an awful job to do.

PN: Do you think the ecosystem, the TEX ecosystem needs a revolution to stay relevant or do you think that this is going to exist forever?

PK: Well, I think there's there's pretty much a good underground movement that requires a certain level of typesetting, and there always will be. And that, so it's not going to go away.

From my point of view, the thing that's really made a difference is things like Overleaf and ShareLATEX, historically. You need GUIs. I mean, you just, you have to make it appealing to people with GUIs. And that's the way the world is now, and has been for some time. So, in general terms, I think you're going to need to hide code from people. I mean, that's just the way everything goes now.

But, in terms of academic publishing and specialist publishing, which will always be there, then I think it's always going to be relevant. Because you just can't do certain things outside of that ecosystem. It's incredibly difficult to do decent typesetting in any of the standard High Street packages for any of this stuff. It's just awful and, and...

So I don't see it's going to particularly grow until you can just isolate people from seeing any code at all, because the last couple of generations of kids, I mean, they're used to apps. And there, you're just tapping stuff, tapping bright colorful buttons, and that's not going to change for generations. So I don't see any particular growth there.

One thing that would make a big difference, which I'm not seeing a lot of movements on but I'm not particularly familiar with what's going on there, in terms of journal publishing, academic publishing, when I know a lot of technical journals, for example, use TEX workflows, but for bibliography management, a lot of those are based on BIBTEX and that's partly because it's very hard to change, you know, historically complicated pipelines for journal publishing. But also there's a technical limitation with LATEX because when you run BIBTEX, you get a file that is the typeset representation of your bibliography. Your `.bbl` is just a document and you include it. And that's it. That's the typeset bibliography.

BIBLATEX does this completely differently, and the `.bbl` that you get out of it is not a typesettable document. It's effectively a TEX macro database of your bibliography and then you have to still apply a style to that during the final PDF output. And that makes it an awful lot more flexible, and you can do things you can't do in BIBTEX, but it also makes it very difficult for people, for journals to implement in their pipeline when they want someone to just send them the typeset bibliography. There are hacks to sort of do this I've been kicking around, but there's no easy way of doing this.

Your bibliography is not a standalone typesettable file. It's all pulled in from, you know, various bits and pieces during the final processing.

PN: So that, that's what, that's what keeps the publishers away from BIBLATEX?

PK: It certainly will keep some of them away, yes, because when you submit the bibliography they just want a typesettable file that's using one of their `.bst` files, and there's really no easy way of doing that with BibLaTeX unfortunately, unless they redid their whole pipeline, which for most publishers, given that the margins are so small, particularly academic publishers, that motivation for redoing their entire publishing pipeline is incredibly small.

PN: Do you use TeX to submit your papers?

PK: I write them all in TeX, but I almost always have to convert them from PDF to Word or something horrible in order to send them because they just won't...

If a lot of them are using, particularly for example, Routledge or Blackwell, they use these very quite sophisticated online submission things for journals now, and they're quite nice and they work very well. But you have to send it all in Word. They work natively with Word, and that's it.

PN: How do you feel, putting this work into a document and it's not coming out in the finished product?

PK: Well, I don't think any of us like that. I mean it's a bigger topic, about certain types of cultural phenomena that I often have with, sometimes with patients and often with students in the past that...

There are a certain sort of sophistications and discriminated positions that matter independently of whether they are practical or not. And so it, yes, it would be easier for me to just write all this stuff in Word and just submit it, but... There are some things that, typesetting and making things look nice is intrinsically, it's intrinsically more structured and more developed than not doing it. And so there's an intrinsic value in that, which it doesn't really matter to me whether it's practical or not. I know it sounds a little bit abstract, but I tend to think that's a rather important principle in general.

You shouldn't try to be too efficient in life. There are consequences for doing that. So I'm quite happy with, you know, "wasting" time typesetting something in a really nice way and then having to export it to Word which loses everything. That's all right.

PN: I am a producer of publishers' workflows and for the first time in my life I'm producing a work for a publisher down in Brazil, which uses BibLaTeX.

PK: Aha!

PN: ... exactly because they have problems sorting bibliographies which use Portuguese and they are sick of BibTeX because of the sorting algorithms, and they have this explicitly asked for BibLaTeX and this was my first. I was in fact very, very surprised. I thought they were...

PK: Nice to hear it. I'm glad to hear it.

PN: But is there anything that can be done to ease the path? Because it's really painful to be able to use the facilities of programmable bibliographies and not seeing some of the results later on.

PK: Yes.

PN: Anything that we can do to ease that?

PK: I've thought about it a little bit. I mean, there have been some suggestions I've noticed on StackExchange of trying to, you know, extract the processing of the bibliography out to a file, to a separate file, but this is... I'll be honest, I'm not a particularly brilliant TeX hacker. I think Philipp Lehman, the original author of BibLaTeX, was a lot better at it than I am, and a lot of other people are. I think there have been some comments... I don't know if Joseph Wright is on here but I mean it will require Joseph Wright levels of skill to do that kind of hacking on the underlying page, page dumping algorithms to actually get, you know, something like BibTeX's output from BibLaTeX. I don't think I've got the skill to do that. But maybe we will be able to, you know, have some kind of thought about this, but as I remember, it somewhat depends on some of the LaTeX3 stuff coming out. So, and we've already started to move BibLaTeX toward LaTeX3 a little bit but...

Yeah, that's going to be a bit of a job to get it all over. And once it is, I think there's a few more options in that direction. But right now it's a discussion thread for StackExchange.

Q&A

PN: Well, I'd like to invite people that if they have any questions to ask you to come and join us. I mean, the panelists can do that by themselves, and the attendees all over the world can wave and I can upgrade them to a panelist so that they can ask the question themselves, would you mind that?

PK: That's fine.

PN: So, if you have any questions, just please unmute yourself.

PK: I can say while we're waiting that I did make a vague promise last October at the DANTE talk in Germany that we would be releasing an experimental multi-script version of BibLaTeX and Biber early this year, and support much more multilingual stuff.

So you can have bibliography entries that contain different scripts in the same field, which has been a long requested thing for multilingual use and that now does exist.

PN: OK, I can attest to that. I am the author of a book here in Berkeley called "Berkeley problems in mathematics" which is sort of the entrance exam for the PhD in mathematics. And the book recommends, you know, cites a lot of other books that you should read in order to prepare for this particular exam. And this book exists in many translations, to Vietnamese, Chinese, Korean...

PK: Right, right.

PN: Spanish, sometimes Portuguese, French and German, and then we cite all those. And what I have, the hoops that I have to jump through to do that in BibTeX, are tremendous.

PK: Yeah, it's painful. Well, the whole point of the multilingual stuff is supposed to be to make that possible. We have an experimental branch, it should be, it was an awful job to actually implement it and it should be completely backwards compatible, but it has a whole new syntax in the `.bib` file for multilingual data database entries. So, we have a manual for it and everything. So this is going to be a bit of a push probably early next year.

PN: Thank you. Thank you. There's one question that came up. How are people with social anxiety doing with remote sessions? I guess somebody is trying to get an e-consultation out of you!

PK: Well, it's a good question.

Generally speaking, people with social anxiety do better with remote sessions because they're less social. The remote sessions are a different beast entirely. They are, I generally don't like them that much because the kind of depth psychology I do, it's not, it's not just purely sort of chatting, there's a lot more of what you call tracking, a lot more of a kind of feeling tension in the room and things like transferential considerations which is very difficult to do remotely.

But there also are benefits. I mean, one thing you find, I found when I was training and seeing patients in the US, it was quite funny. One thing you find quite quickly is when you're having a phone conversation, particularly with no video, just a phone conversation with somebody, instead of being in the room. One thing you find is people are prepared to say things they would never, ever say to you face to face. And it's also, to a lesser extent, through video. There are research papers on the fact that, working with social anxiety, doing it remotely, doing it by telephone, it's actually better very often because it's a way to lead people in because there's simply less social signaling going on, and there's just less in the environment to trigger anxiety. So generally quite well is the response to that.

PN: People stay in their own environment.

PK: Right. Yes. And they are not exposed to unpredictability and other environments. And so it's generally not so bad.

PN: Interesting.

This has been quite a nice conversation. I loved the interview and I hope we can do this in person at some point in time.

PK: Yes. When the world stops burning and being crazy. Yes. Maybe.

PN: There is a question by Michael Topp, and he asks, "Philip. I'm going to translate a couple of books and I want to do it with LaTeX, of course, and bibliography, and it's an art book. So is it really hard to sell them to publishers?"

PK: Well, not necessarily. I mean, it just depends on what their workflow is. I mean, I mean some publishers will just be perfectly happy if you provide them with a very nice PDF. And so it doesn't matter how you create that. So then you're fine, it just depends. If they have a LaTeX workflow that you have to somehow fit into, the chances of them using the packages you want are quite small usually, but it just depends what they want. I mean, it's not that hard to sell them... I mean, look at what I do. I mean, I don't even tell them I use LaTeX, they just want some kind of, you know, Word thing at the end. And generally, I just produce a nice PDF and then I just use some sort of Adobe tool that creates, that turns PDF into Word. And while that's a fairly unpleasant thing to do, I mean, I get it to work in the way I want, and then I get to deliver it in the format they want. So it's not so bad.

But it just depends what they do. I mean, no it's not, it's only hard to sell it to people who have very particular requirements and their own `.bst` styles with BibTeX usually. Then it's a bit hard to sell it to them.

PN: Well, on that note, I'd like to thank you very, very much and hope to join you in person, when there's some time for a continuation.

PK: Thank you very much.

◇ Paulo Ney de Souza
tug.org/tug2020

Beyond Roman fonts: Extra dimensions in Malayalam fonts

Hussain KH, Rajeesh KV, Aravind Rajendran

Abstract

The Malayalam alphabet consists of 51 basic characters which combine to form more than 900 conjuncts (ligatures) in traditional script. Unlike Roman scripts, Malayalam has two kinds of ligatures, namely Horizontal and Vertical conjuncts. Vertical conjuncts in Malayalam are unusual among other Indic scripts, and almost never seen in Roman scripts. Anatomically there are two parts in vertical conjuncts — Above Character and Below Character.

Vertical conjuncts demand careful attention in space management in Malayalam types. Accommodating the below character in the below-base(line) space when adhering to Roman metrics poses serious inconveniences. Compressing all levels into a single level harms the shapes of around 700 vertical conjuncts, as Malayalam poses extreme cases of divergence while dealing with vertical conjuncts. Designing vertical conjuncts results in many deviations from the accepted norms of Roman typography. Deviating from Roman metrics poses problems of point sizes when typesetting documents using Malayalam and Latin text together.

Thus, creating original Malayalam script fonts while satisfying dimensions of Roman fonts exerts formidable pressure on the designers. Malayalam typography looks at geometrical consistency in a different way than Roman typography.

1 Malayalam script and Rachana

Malayalam, the mother tongue of 45 million people in Kerala, the southernmost state in India, is one of the 22 official languages of India. It is a 1600-year-old Dravidian language, and its script is classified as abugida, or alphasyllabary. Unicode support for Malayalam script was in GNU/Linux systems around 2002 and MS Windows XP added support in 2004. Since then Malayalam language technology has seen significant advances, thanks to Swathanthra Malayalam Computing (`http://www.smc.org.in`) and its Rachana font based on the traditional script.

Figure 1: Ligatures

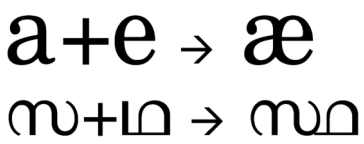
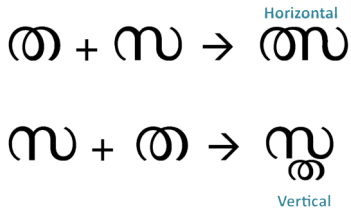


Figure 2: Horizontal and vertical conjuncts.



Unicode encodes the basic characters in Malayalam [1] while all the complex conjunct characters are supported by OpenType shaping rules (see Appendix A for information about the glyphs in the Rachana font). The Malayalam script also requires 'complex text shaping' support from shaping engines (notably HarfBuzz, used by XeTeX and (as of 2020) LuaLaTeX) to properly shape its conjuncts and vowel symbol combinations.

1.1 Old vs. New script

'New lipi' or Reformed script of Malayalam, which tried to simplify the original script by breaking many conjuncts and vowel combinations, appeared in the 1970s as an official government effort to use Malayalam with English typewriters and Linotype typesetting. ASCII fonts with 140 characters based on the New script were later popularized by desktop publishing (DTP). New script is still in vogue in typesetting, though it is only a subset of the original script with 900 conjuncts standardized for printing by Benjamin Bailey in 1824. New lipi with its limited conjuncts can easily be accommodated in fonts with Roman font metrics/dimensions.

1.2 Rachana

The language campaign named 'Rachana' in 1999 began advocating for use of the traditional script in Malayalam computing. After the advent of Unicode Malayalam in 2002, the traditional script (Original Script), popularly known as 'Old Script', has had a steadily growing presence in the Web and in typesetting and printing. The traditional orthography demands more space beneath the baseline for vertical conjuncts. Designing fonts for original script

Figure 3: Metrics of below base character.

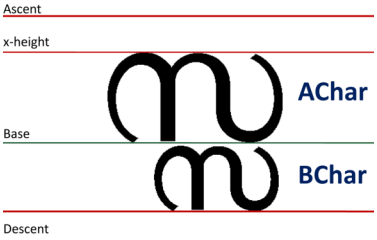
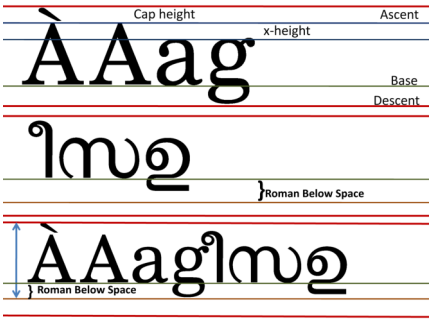


Figure 4: Below-base parts and space in Roman and Malayalam (basic characters) type metrics.



satisfying dimensions of Roman fonts exerts a lot of pressure on the designers.

In this discussion typography of Original Script (Old Lipi) is considered, since it is a superset of all variations of New Lipi that exist in various fonts used for DTP. Typesetting is now shifting from New Lipi to Old Lipi, thanks to growing usage of Unicode.

As mentioned, the Malayalam alphabet consists of 51 basic characters which combine to form more than 900 conjuncts (ligatures) in traditional script. The Rachana font, first designed in 1999 (so-called ASCII[1]) for the campaign for traditional script, underwent major modifications in 2000 and especially in 2006 when the font became Unicode-compliant and distributed under the GNU GPL. Now, after a long period of 15 years, all 1000+ glyphs in Rachana have been totally redesigned, taking more graphical liberties with fewer constraints exerted by Roman typography. This paper explores new possibilities in Malayalam typography and how far beyond Roman typography it can go without compromising either aesthetics or functionality.

[1] Before the advent of Unicode, Indic script fonts followed an encoding similar to that of Latin scripts by laying out characters in an 8-bit table, known as ISCII. To accommodate the 900+ characters needed, Rachana was originally designed as a set of six fonts. The typesetter then manually switches fonts in a DTP program to pick specific characters.

Figure 5: Beyond Roman type metrics.

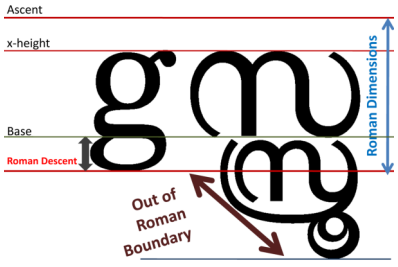
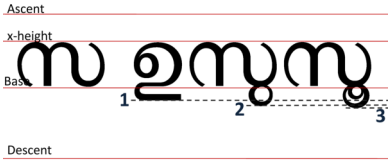
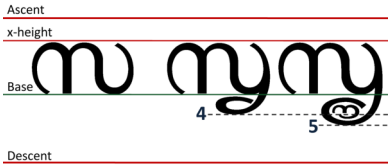


Figure 6: Below-base levels 1–5.

(a) Below-base levels 1–3.



(b) Below-base levels 4 and 5.



2 Ligatures/Conjuncts

Similar to Roman types, Malayalam also has ligatures, known as conjuncts. They are formed by combining basic characters (see an example in Fig. 1). The considerations of moving away from Roman typography mainly relate to conjuncts, which amount to nearly 20 times the number of basic characters.

2.1 Conjuncts: horizontal and vertical

Differing from Roman fonts, conjuncts in Malayalam are formed in two ways — horizontal and vertical. For example, Basic characters ത (tha) and സ (sa) combine horizontally to form ത്സ (thsa) i.e., ത + സ → ത്സ, whereas സ and ത combine vertically, സ + ത → സ്ത (stha). See Fig. 2 for a depiction of this.

2.2 Below-base characters

2.2.1 AChar and BChar

Vertical conjuncts demand careful attention in space management of Malayalam types. Anatomically, there are two parts in vertical conjuncts — Above

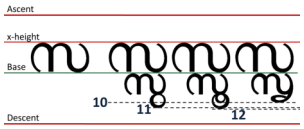
Figure 7: Below-base levels 6–9.

(a) Below-base levels 6–8.

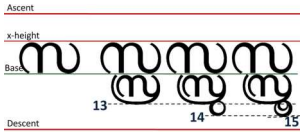
(b) Below-base level 9.

Figure 8: Below-base levels 10–18.

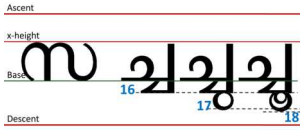
(a) Below-base levels 10–12.



(b) Below-base levels 13–15.



(c) Below-base levels 16–18.



Character (AChar) and Below Character (BChar). While designing glyphs, AChars are placed above the baseline, filling the x-height. BChars are placed below the baseline and are always smaller in size than AChars, following the pattern in handwriting and calligraphy (see Fig. 3).

2.2.2 Below-base space

Parts of glyphs going below the baseline occurs in Roman glyphs, e.g., the letters 'g' and 'j'. This happens in Malayalam as well and its basic characters (adopted in the Malayalam Unicode chart) are fit well in the above and below spaces allocated as in normal Roman types (see Fig. 4).

3 Objective of the paper

When it comes to vertical conjuncts in Malayalam, accommodating BChars in the below-base space allocated in Roman metrics poses significant problems. The peculiar behaviour of 'space grabbing' of vertical conjuncts, as we'll see, is in perpetual collision with the Roman below-base space and BChars often go below the Roman descent boundary (an example is in Fig. 5). The main objective of the paper is to show how this is circumvented in the typography of Rachana, going beyond Roman metrics.

Figure 9: Extra descent in Rachana.

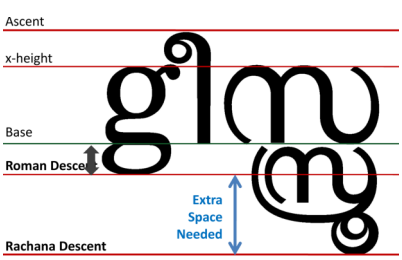


Figure 10: Below base space: varying vs. constant.

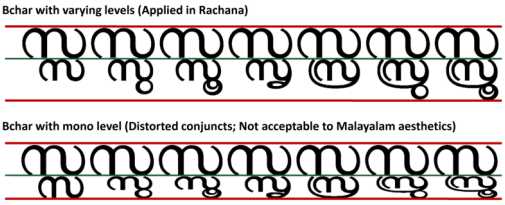


Figure 11: Equalizing levels 1, 4 and 6.

3.1 Levels in below-base

The large vertical space demanded by BChars below the baseline is a challenge in Malayalam typography which in principal should be specially treated for appropriate leading (interline space). Owing to consonant-vowel pairing, BChars have 18 different heights/levels, explored in Figs. 6, 7 and 8.

3.2 Treating BChar levels

From Figs. 6a to 8c, it is evident that the vertical space between baseline and descent allotted in normal Roman fonts is too little to accommodate BChars. Malayalam glyphs require nearly the same space above and below the baseline (shown in Fig. 9). Below-base space cannot be compensated by taking from the Above-base space since some of the Malayalam vowel signs use the full cap-height; i.e., space above the base is needed exactly as in Roman fonts. Below-base space allocations in Roman type is woefully inadequate for Malayalam, leading to some unfortunate treatments in designing vertical conjuncts.

One of the solutions for accommodating BChars in Roman-descent is to squeeze all levels into a single level. This is graphically possible but produces distorted heterogeneous characters totally unacceptable to Malayalam aesthetics. In fact this kind of deformative practice is unacceptable to the typography of any script in the world. For instance, see the single-level BChars in Fig. 10.

Figure 12: Equalizing levels 7–9.

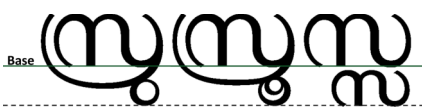


Figure 13: Equalizing below-base levels 10 and 13.

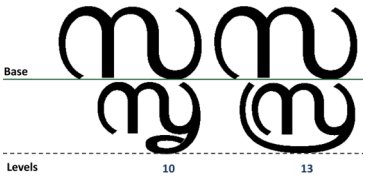
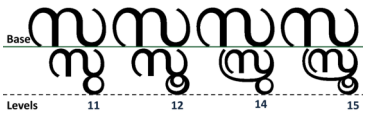


Figure 14: Equalizing below-base levels 11, 12, 14 and 15.



3.2.1 Equalizing near heights

Compressing all levels into one obviously harms the shapes of most vertical conjuncts in Malayalam. Reducing the number of levels can be achieved by another method. In earlier versions of Rachana, attempts were made to reduce vertical levels by equalizing near-heights, which reduced the number of vertical levels from 18 to 6. By this it was hoped to attain almost evenly sized BChars in vertical conjuncts. It could be a potential way to contain the unusual leading in Malayalam typesetting caused by lacunae in vertical conjuncts (Figs. 11, 12, 13 and 14).

3.3 Typographic deviations of BChar

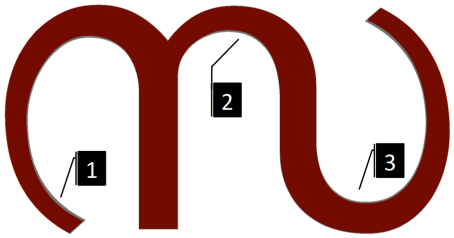
Even though vertical conjuncts have two parts, it must be remembered that both are integral parts of a single character. Naturally one may expect both parts to follow the same typographic characteristics, but in reality they differ (rather, are forced to differ). Unfortunately, it is found to be impossible to keep the same types in above and below parts. The practice adopted in designing Rachana results in violations of basic rules of Roman typography (see Fig. 15).

Let us consider creating the vertical conjunct ന്ന which consists of the same character ന in above and below parts.

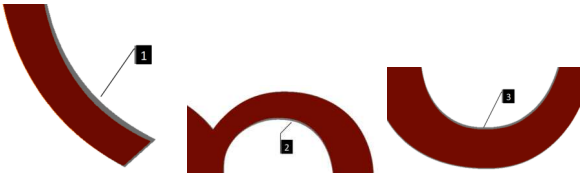
Figure 15: Typographic deviations of single conjunct ന്ന.

Figure 16: Typographic deviation of same character ന in the conjunct ന്ന.

(a) Superimposed AChar (red) and BChar (gray). Notice the curves marked with 1, 2 and 3.



(b) Enlarged parts of Fig. 16a.



As seen in Fig. 15(1), when a same-sized AChar is placed in the below-base area, the conjunct produced is strikingly disproportional. The same shape with the same size produces an optical illusion of an oversized BChar. A BChar should invariably be smaller than the AChar in Malayalam orthography.

In Fig. 15(2), a 60% uniformly scaled BChar strictly adheres to the type design of AChar, but produces an unbalanced shape. The small BChar appears to suffer from pressure under the big AChar.

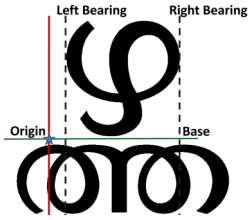
Fig. 15(3) shows a more balanced shape, achieved with non-uniform scaling. This BChar is designed with 70% horizontal and 60% vertical scaling. It produces a more pleasing effect compared to uniform scaling, at the same time not increasing the vertical size. More or less this proportion is followed in all versions of Rachana. Here a wider BChar 'supports' its counterpart in AChar and helps to achieve legibility at lower point sizes (10pt or 11pt) while typesetting.

Please observe, non-uniform scaling of BChar produces a *different type*! This can be verified by superimposing a same-sized AChar and BChar. As seen in Figs. 16a and 16b, the curve of BChar (light gray in colour) often varies from AChar (red online, dark gray in print) due to non-uniform scaling. This

Figure 17: An extreme case of below base conjunct.

Figure 18: Scaling adjustments for glyph in Fig. 17.

(a) BChar with 70%–60% scaling.



(b) BChar with 45%–50% scaling improves intercharacter spacing.

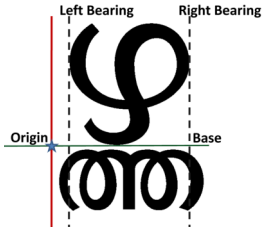


Figure 19: Design adjustments for an extreme case of BChar.

(a) Collision.

(b) Larger intercharacter space.

(c) Unconventional reduction.

is a clear instance of typographic deviation between AChar and BChar.

This kind of typography with two kinds of types in the same glyph is almost unheard of in Roman types. Rachana takes the liberty to deviate from these accepted norms.

3.3.1 Extreme cases

Some vertical conjuncts are shaped deviating more from the usual 70%–60% proportion, depicted in Fig. 17.

If 70%–60% scaling is applied, the BChar extends far beyond the left bearing and right bearing, resulting in collisions with neighbour characters (Fig. 18a). If kerning is adjusted to avoid this, white space to the left and right of AChar produces a 'space effect' (Fig. 19b).

The only solution for these extreme cases is to apply a different proportion to BChar. In Rachana, 45%–50% scaling instead of the usual 70%–60% is applied in designing these types of vertical conjuncts to preserve normal character spacing (see Fig. 18b). This kind of elasticity applied in Malayalam breaks all established rules of typography. It perhaps does not occur even in typography of other Indic scripts.

All these considerations show that Malayalam fonts cannot be designed according to the metric calculations of Roman typography. This is more or less the case with all Indic scripts, due to the abundance of conjuncts. Malayalam poses extreme cases of divergence while dealing with vertical conjuncts. Different proportions applied to different conjunct formations in the same font completely contravenes principles formulated for Roman text types; however, these complexities are perhaps comparable to Roman typography employed in typesetting mathematics (see Fig. 20). There too, different types in a single set are not tolerated.

3.3.2 A page typeset in Rachana

The Rachana font is reimagined and redesigned using many levels of descent, and yet this doesn't cause serious issues with leading. The overall aesthetics and readability are in fact improved. A sample document typeset using XeTeX is shown in Fig. 21.

4 Conclusions

Vertical conjuncts in Malayalam are unique compared to Roman scripts and other Indic scripts. Designing vertical conjuncts results in many deviations from accepted norms of Roman typography. Even

Figure 20: Typesetting mathematics using the Monotype 4-line system. *Source: Daniel Rhatigan.*

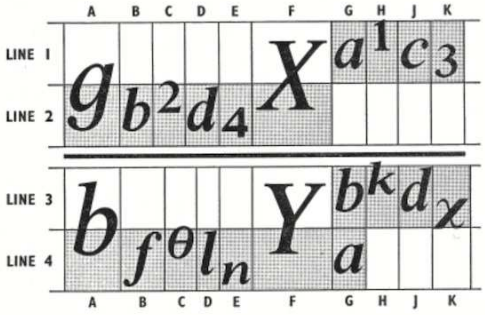


Figure 21: Malayalam text typeset with Rachana.

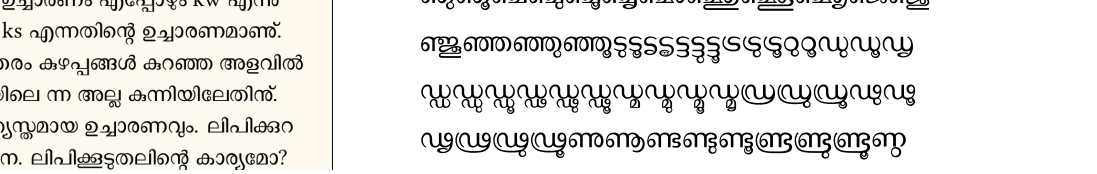
മലയാളികൾ എഴുതുംപോലെ ഉച്ചരിക്കുന്നവരാണെന്നും എല്ലാ ഉച്ചാരണത്തിനും നമുക്ക് എഴുത്തുണ്ട് എന്നും തീരെ തെറ്റായ ഒരു ധാരണ പലരും പറഞ്ഞു പരത്തുന്നുണ്ട്. ഇത്തരം ഒരേർപ്പാട് ഒരു ഭാഷയ്ക്കും ഇല്ല, ആവശ്യവും ഇല്ല. എഴുതിക്കണ്ടാൽ ഉച്ചാരണം തിരിച്ചറിയണം ഉച്ചരിക്കുന്നതു കേട്ടാൽ എഴുതാനും ആകണം—അത്രമതി. അതിനെന്തു വേണം? പ്രസക്തമായ അർഥവ്യാവർത്തനങ്ങൾ എഴുതിക്കാൻ ഒത്ത ലിപി വേണം. പ്രസക്തമായ അർഥവ്യാവർത്തനങ്ങൾ എഴുതിക്കാട്ടാൻ ലിപി ഇല്ലാത്തതാണ് ഇംഗ്ലീഷിലെ എഴുത്തുരീതിയുടെ ഒരു കുഴപ്പം. ഒറ്റസ്വരങ്ങളും ഇരട്ടസ്വരങ്ങളും ചേർന്ന് ഇംഗ്ലീഷിൽ 20 എണ്ണമുള്ളതായി ഓക്സ്ഫഡ് ലേണേഴ്സ് ഡിക്ഷനറി പറയുന്നു. ലിപികളോ a, e, i, o, u എന്ന് അഞ്ചെണ്ണം മാത്രം. ആവശ്യത്തിന് ലിപിയില്ലാത്തതുപോലെ വേണ്ടതിലേറെ ലിപികൾ ഉണ്ടതാനും. Q എപ്പോഴും u ചേർന്നേ വരൂ. Qu എന്നതിന് ഉച്ചാരണം എപ്പോഴും kw എന്നു മാത്രം. X എന്നതിന് ks എന്നതിന്റെ ഉച്ചാരണമാണ്. മലയാളത്തിലും ഇത്തരം കുഴപ്പങ്ങൾ കുറഞ്ഞ അളവിൽ ഇല്ലായ്മയില്ല. കന്നിയിലെ ന്ന അല്ല കുന്നിയിലേതിന്. ഒരേ ലിപി, രണ്ടു വ്യത്യസ്തമായ ഉച്ചാരണവും. ലിപിക്കുറവിന്റെ കാര്യം ഇങ്ങനെ. ലിപിക്കുടുതലിന്റെ കാര്യമോ?

within a font, difficult situations arise when following a single type or rule.

'Leading' in Malayalam typesetting is a serious concern, especially when using fonts like Rachana based on traditional orthography. Leading should be kept as small as possible, and as a result it exerts pressure on Malayalam font designers due to abundant height variations in BChars. Holding a fixed proportion in BChar hurts leading. The 'negotiations and adjustments' in proportion are often applied in BChars but this results in different calculations for ascent–descent and point size estimates. That is why Malayalam font makers are often forced to explore different metrics. Other Indic scripts with moderate below characters/diacritical marks are luckier in this aspect. Deviating from Roman dimensions poses problems of point sizes when typesetting documents using Malayalam and Latin fonts together.

Rachana in earlier versions attempted to group and equalize many below-base levels but later promoted natural proportions in shapes rather than 'forced' proportions. For all intents and purposes Malayalam typography deviates from geometrical consistency set for Roman typography. Proper balance in glyph composition in Malayalam can only be achieved by embracing a different mindset, going beyond Roman typography.

Figure 22: Sample glyphs of Rachana.



A Glyphs in the Rachana traditional orthography font

A sample of the glyph set designed for the Rachana font is shown in Fig. 22. The complete list, which contains the comprehensive character set for traditional orthography is available at `http://rachana.org.in/docs/Rachana-conjuncts.pdf`.

References

[1] Unicode Consortium. Malayalam Unicode chart, Unicode standard 13.0 [online]. `https://www.unicode.org/charts/PDF/U0D00.pdf`

◇ Hussain KH
Rachana Institute of Typography
`hussain (at) rachana dot org dot in`
`http://rachana.org.in/`

◇ Rajeesh KV
Rachana Institute of Typography
`rajeesh (at) rachana dot org dot in`
`http://rachana.org.in/`

◇ Aravind Rajendran
STM Document Engineering
`aravind (at) stmdocs dot in`
`https://stmdocs.in/`

The road to Noto

Steven Matteson

Editor's note: This is a lightly edited transcript of the talk given at the TUG 2020 conference. Some of the illustrations are omitted here; for the full set, and the video of the talk, see `tug.org/tug2020`.

The Noto family of fonts is one of the largest undertakings in the history of type founding. It certainly has not been a straight line from point A to B. I've been involved on and off for 14 years, and there are about 60 others who have contributed to it up to this day. This doesn't include the efforts on the Chinese, Japanese and Korean fonts, which people from Adobe would have to tell you about.

For the purposes of this talk the road to Noto begins with the Rosetta Stone:

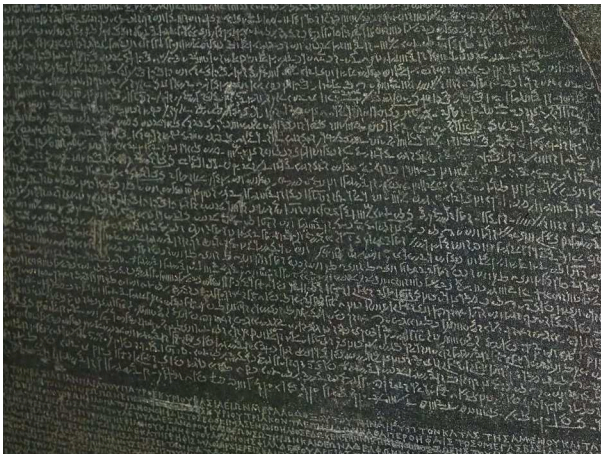


Figure 1: The Rosetta Stone, 196 BCE.

a 3.5-foot tablet fragment, similar to granite, with the remarkable workings of hand and chisel, carefully spelling out an imperial decree in three different writing systems. The stone's historical significance is legendary. I clearly remember studying it in seventh grade World History. We had to make our own version out of modeling clay and mark it with our own messages with a toothpick. We even had a contest to try and translate each other's messages. Thank you Oak Park, Illinois Public Schools.

The markings are King Ptolemy V's "Memphis Decree", given in 196 BCE during turbulent political times and cultural upheaval. The Rosetta Stone is a fascinating example of the painstaking efforts made to produce a document in multiple languages — in this case Hieroglyphs, Demotic Script and Ancient Greek — all with the aim that multiple cultures and generations understand this single message.

Fast forward to 1573 to Christophe Plantin's Polyglot Bible printed in Antwerp and funded by King Philip II of Spain.

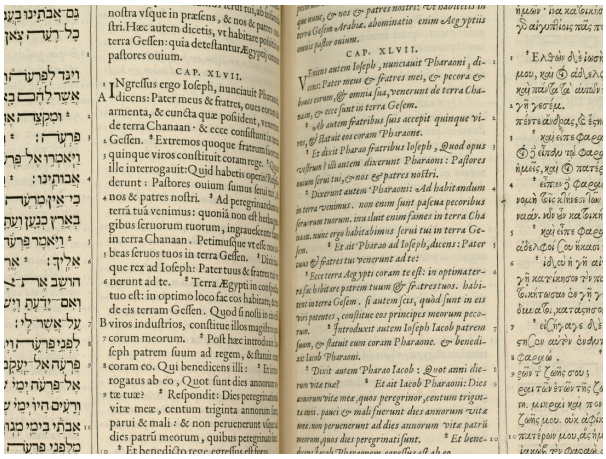


Figure 2: Christophe Plantin's polyglot Bible, 1573.

It was printed in six volumes of the different books of scripture and two additional volumes which contained translation dictionaries to accompany them. The text is translated into Hebrew, Greek, Aramaic, Syriac and Latin texts. The typography is stunning in its beauty, simplicity and painstaking planning.

The spread above shows Hebrew script on the far left with Latin in an upright roman typeface. The far right is a beautiful flowing cursive-looking Greek text; the accompanying Latin is in italic to complement the look and texture of the Greek. This typographic detail helps unify and bring harmony to the page despite the differences in the multilingual writing systems.

For me the achievement here, the complexity of the formatting and quality of printing, is inconceivably beautiful.

Starting around 1654, about 70 years later, in England, Bishop Brian Walton began work on his polyglot bible (next page). He published *nine* translations — Aramaic, Hebrew, Syriac, Arabic, Samaritan, Ethiopic, Greek and Latin. This production was funded by subscription rather than a grant from a government or church body. In just a year the bishop had found 400 private contributors anticipating the finished book.

Some consider this the least beautiful of all the polyglot bibles. It may be because it was not a royally funded project with commensurate royal flourish. Or maybe because of the complexity of nine translations vs. the four or five of previous works, which challenges the typographer to assemble a harmonized page.

Figure 3: Bishop Brian Walton's polyglot bible, 1654.

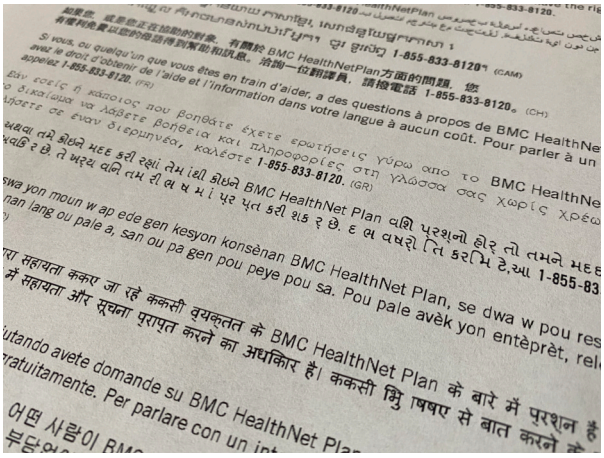


Figure 4: A common contemporary polyglot page.

But for me, Walton's achievement, the complexity of the formatting and quality of printing, is inconceivably beautiful. Particularly compared to where we are 400 years later with a standard insurance company's explanation of benefits statement (above).

This polyglot page is now commonplace and easy to do with our current typesetting tools. However, like many generic or institutional forms we see day to day, this page can be vastly improved upon. The line lengths are excessive for most of the text represented here. The variety of type styles makes it appear to be a ransom note rather than a serious document. The boldness of some of the translations makes them appear far more important than other languages, so not very egalitarian. And, from a branding standpoint, the visual identity of this company is not maintained.

I don't want to negate the complexity happening behind the scenes to make this page possible. It is far more intense than the reader will ever, or should ever, know or worry about. Just the ability to shape text right to left was a big step in computing, let alone the other magic going on here to typeset in all these languages.

Multilingual, or polyglot, typographic pages can get worse than this. An author's worst nightmare might be for his or her reader to come upon a page that looks like this:

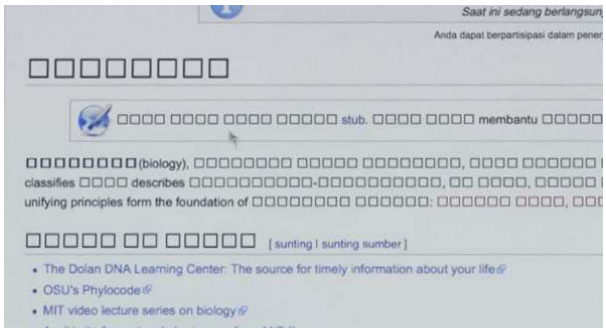


Figure 5: Tofu.

This page is full of missing glyphs. I received this image from Google's Bob Jung, who orchestrated much of the early part of the Noto project on Google's side. If a computer system is missing a character that was entered by the author, the reader sees the undefined glyph — typically an empty square. As Bob told me, "The squares remind a lot of people, particularly in Asia, of packaged bean curd. Tofu."

And that's where the Noto fonts got their name: No To(fu) = Noto.

My road to Noto began in 1985 when I started at RIT's school of printing. In my typography classes I was introduced to hot metal typesetting juxtaposed with the latest computer typesetting equipment of the day, i.e., bitmap fonts.

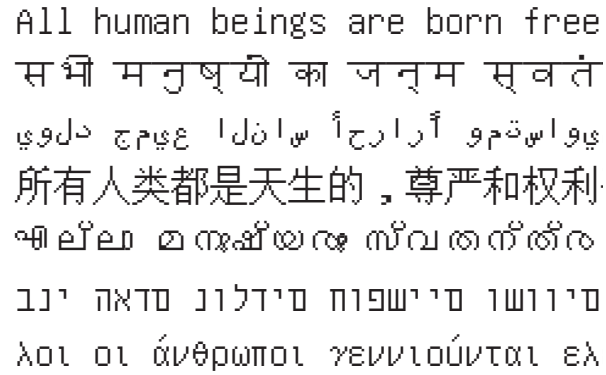


Figure 6: My road to Noto starts, 1985.

Prof. Archie Provan was working as a consultant to Xerox on their efforts to gather bitmap fonts from many different foundries that would be able to typeset all the world's languages. He was working

with Ed Smura on the AFII standard — the Association for Font Information Interchange. This included other aspects related to typography, including typeface classifications, but the bitmap font project was probably the most ambitious aspect.

With these bitmap fonts, the Xerox Star publishing system, a precursor to the Macintosh, was to be able to produce documents in any language. The pipe dream was for the fonts to be beamed via satellite to a Xerox Star installed anywhere in the world.

When describing the project to me, Archie gave a romantic notion about how, if people around the world could communicate more accurately and easily, they might spend less time fighting.

Homely as they were, these monochrome bitmap shapes would carry the power of our own messages, our own 'Memphis decrees', all around the world.

By the time I graduated, outline vector fonts that we use today were becoming more viable. Single-sized jaggy bitmap shapes were suddenly no longer sufficient when you could infinitely scale a letterform for more expressive typography.

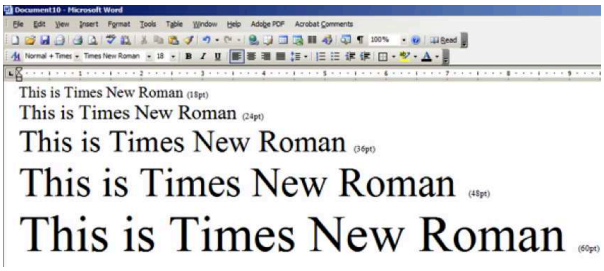


Figure 7: Early outline fonts.

I began working on the TrueType system fonts for Microsoft in 1990. The fonts all had a modest character set of around 300 characters, already more than earlier font formats could handle. TrueType fonts could handle more characters, and thereby typeset more languages, because Unicode became the standard way of encoding or 'organizing' all the letters in all the alphabets in the world, allowing (at that time) 65,536 characters in all. The previous encoding schemes, including such as ISO 8859, typically allowed only 256 characters in a single file.

With Unicode every character in the world gets a unique identifier. For example, the G-breve for Turkish gets a name and a Unicode number (Ğ, U+011E); as does the Greek Omega (Ω, U+03A9), etc. The Unicode registry is constantly being updated. In 1999, for example, Unicode consortium scrambled to put the Euro symbol (€, U+20AC) into its directory so font foundries like Monotype could update their massive font libraries to support the new currency.

Between 1990 and 93, foundries continued developing a steady stream of Unicode-encoded fonts, ever growing in size. The WGL (or Windows Glyph List) character set, defined by Microsoft, raising expectations for fonts to having support for about 90 languages with around 600 unique characters in a font file.

Bigelow and Holmes built their Lucida Sans Unicode font to coincide with the publication of Unicode's 1.0 specification. Lucida Sans Unicode was released by Microsoft in 1993 and added Greek, Cyrillic and Hebrew to the Latin Character set. The font also included support for phonetic and math symbols which Unicode had defined for version 1.0.

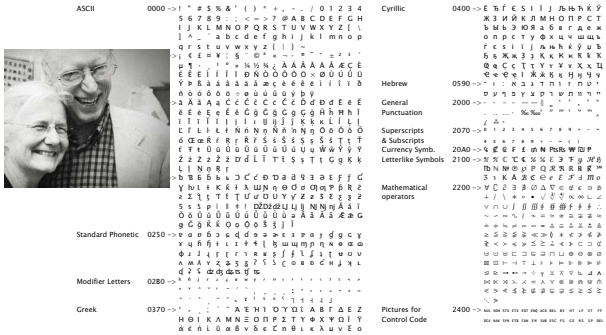


Figure 8: Bigelow&Holmes Lucida Sans Unicode, 1993.

In 1997 Monotype was tasked by Microsoft to extend Arial to cover all of Unicode 2.0 — a mere 50,000 letterforms weighing in at 22Mb for the single font file. While the TrueType font format could theoretically support all of these characters in a single font file, it was a tricky process to make it work. We had to build many small 'fontlets' and then stitch them all together at the end of the process. My colleague of many years, Kamal Mansour, saw to it that the design was, as much as could be expected at that time, harmonious with Arial.

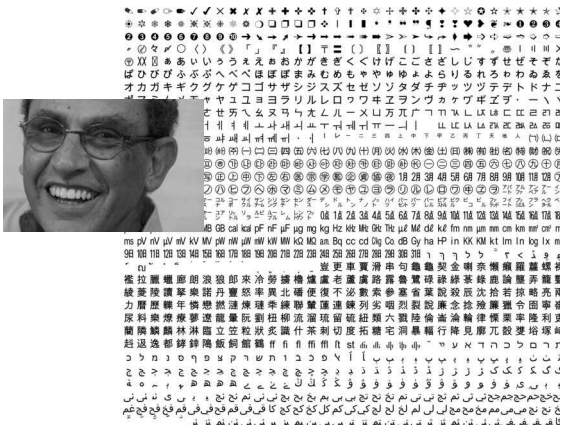


Figure 9: Monotype Arial for Unicode 2.0, 1997.

We included many more Chinese ideographs than were defined by Unicode. The extra 'glyphs' were included to support both simplified and traditional Chinese, so the actual number of letterforms in the font exceeded that of the Unicode standard.

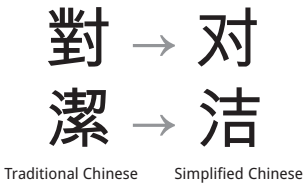


Figure 10: Same Unicode, different shapes.

The illustration above shows how the same Unicode character can be represented by two different glyph shapes. A Western equivalent might be to show the letter g in both a single loop and double loop form — it's the same Unicode character represented by two different glyph shapes.

In 2006 Google approached me to create a typeface family for a new mobile phone platform. Google wanted a unique UI experience for branding Android. A unique interface experience starts, of course, with the kind of typeface you interact with.

Because of Google's and Android's somewhat quirky branding I needed to draw an approachable typeface that was 'left of neutral'. Being too neutral wouldn't stand out as being unique to the brand. But if it was too cute or techno-looking the legibility and functionality would suffer.

Figure 11: New typeface design goal for Android.

Also, Android has a very specific rendering environment. Recall that cell phone screens were nowhere near the resolution of today's devices.

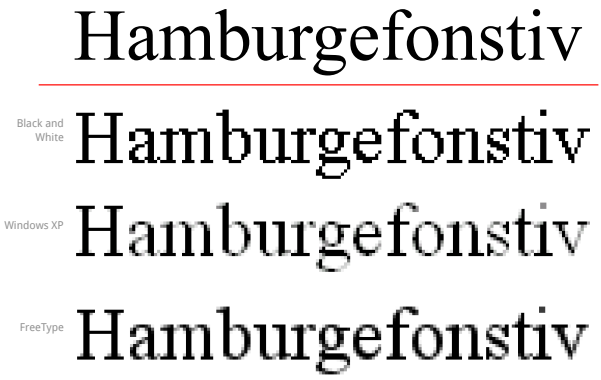


Figure 12: Different screens, different results, 2006.

This next example shows how the same design can be very different looking depending on the screen it's being viewed on. The thin parts of letters can look like they are disappearing or slightly too heavy, depending on how the software interprets and draws the letters.

These are some early drawings and experiments which were put into testing.

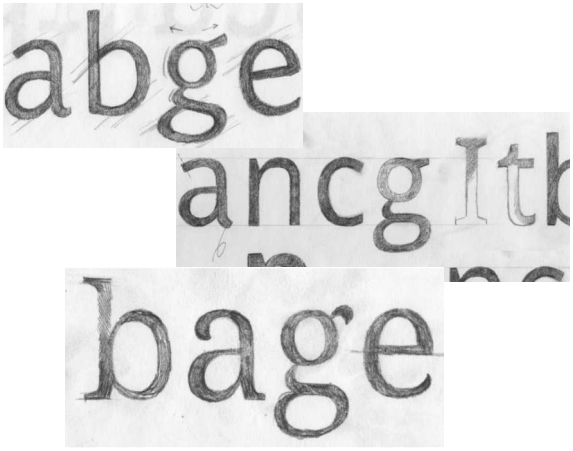


Figure 13: Early drawings for the Android font.

We had to create fonts and install them into devices to view the effects of small changes in design and proportion of individual letters. This was a laborious process because the user interface was being designed at the same time as the typeface was being developed. To complicate things further, the hardware which would become the first Android phones was also in the process of being designed and manufactured.

I designed the fonts with an eye on how they would render in various Android screens. I worked back and forth with their UI team to make sure there was enough contrast between regular and bold weights to aid in establishing a hierarchy in the interface. I made sure that detailing in each design was working well at these limited resolutions.

Once we were on the right track with the design we did some weight tests to see how much contrast was necessary between the regular and the bold weights.

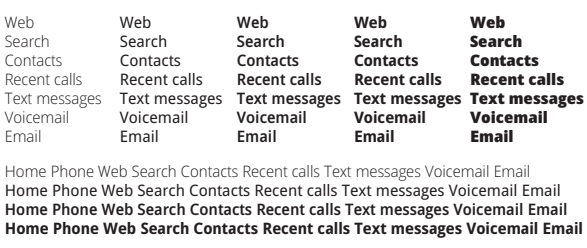


Figure 14: Weight tests for the Android font.

Only two weights were required; italics would be synthesized by Android in order to save storage space. On the other hand, after much discussion it was determined that a serif typeface should be part of the basic set of fonts. The serif fonts would be for reading news feeds and extended text. The sans would be for UI elements and menus.

Figure 15: Original Droid font family.

All of the fonts supported the WGL-4 character set but there was a sense that this was going to be expanded on if Android was successful.

The goal of course was to create a family of fonts which held up at small screen sizes and gave the platform an approachable, friendly appearance. We may have actually achieved this as a writer for Wired's online edition called the fonts 'googly'.

Figure 16: Droid display test.

When the time came to expand on what could be displayed in the Android UI, Google thought it was clear they did not want it to look like the left side below, regrettably similar to the insurance benefits statement shown earlier (fig. 4). Rather, they wanted a harmonized 'Android brand' look and feel across all the languages they were supporting. All of the scripts should have a contemporary, approachable and ergonomic feeling, closer to the right side. (Apologies for the typo in the Arabic text.)

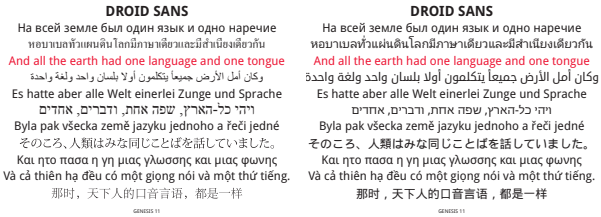


Figure 17: Unharmonized scripts on left; harmonized on right.

Making harmonized designs for scripts which have no historical relationship to each other is a bit tricky and in some cases nothing but a compromise of making things roughly the same weight. In Arabic, for example, the weight is on the horizontals rather than the vertical stems like Latin. This alone makes an enormous difference in balancing the weights. Visual cues can be picked up from the Latin shapes — the soft terminals and weight of the thin protrusions and the general contrast of thick to thin can be harmonized.

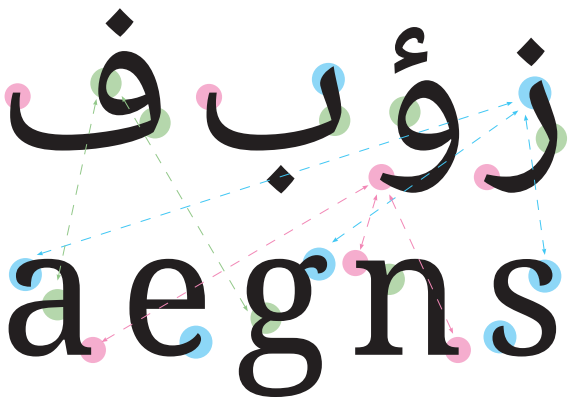


Figure 18: Harmonizations between Arabic and Latin.

The Arabic fonts were designed by Pascal Zoghbi with some art direction from me. The style that best matches Latin serif types is called Naskh. In most Naskh typefaces the counters are tiny, but in this case we exaggerated their size to mimic the Latin type's openness and aid legibility on screen.

In the example below, the top line of Arabic is in a style called Kufi which complements a Latin sans serif more closely. Typically, however, Arabic readers prefer the Naskh style (bottom line) for extended reading. I feel that it's similar to the resistance Western readers used to have for reading books set in sans serif typefaces. It really wasn't until the 1950s and 1960s that people started accepting this new typographic approach.

Working on this project I learned that Arabic readers were very accustomed to having to pinch-zoom text as soon as a page loaded. One of the goals

of exaggerating the proportions was to help prevent the need to zoom in to read default Arabic sizes. My understanding from Google was we achieved this in Droid and Noto.

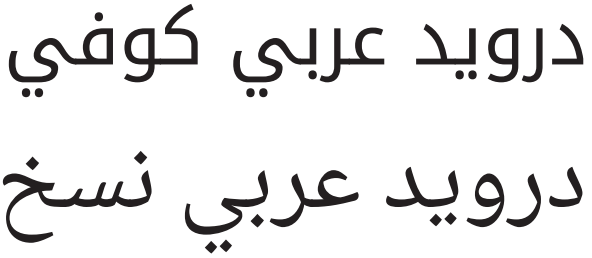


Figure 19: Kufi style (top), Naskh style (bottom).

Beyond Arabic, a Thai design was another early need for Android. This is serif style and it can be noted where some of the details are hinting toward the Latin serif typeface. (This and following examples are truncated on the left and/or right so details can be better seen.)

Figure 20: Droid Thai, with Latin for comparison.

A sample of the Droid Serif Hebrew — another script where the challenge is in the weight distribution being opposite that of the Latin.

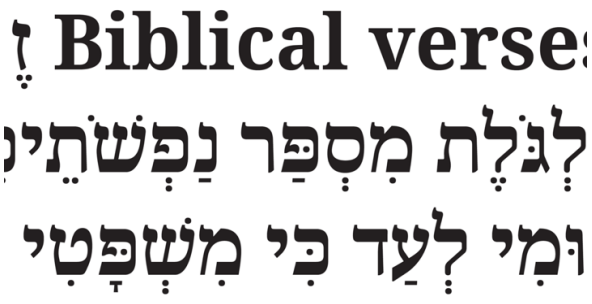


Figure 21: Droid Hebrew.

The Ethiopic script is often seen in a slanted form similar to an italic. I decided that an upright form would be most legible and useful for Android's user interface.

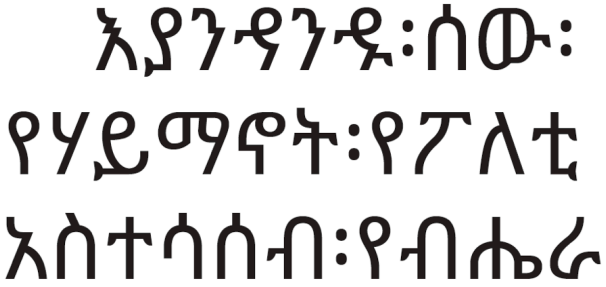


Figure 22: Droid Ethiopic.

Armenian takes many cues from the Latin lowercase shapes making it considerably easier to harmonize.

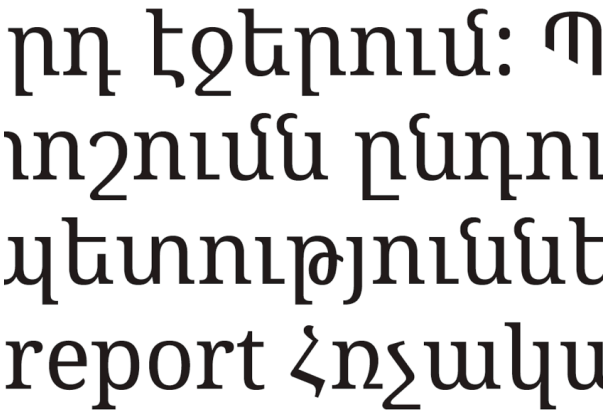


Figure 23: Droid Armenian.

Similarly, Georgian takes many cues from the round shapes found in the Latin. An entirely different texture than Armenian, but clearly a member of the Droid typeface family.

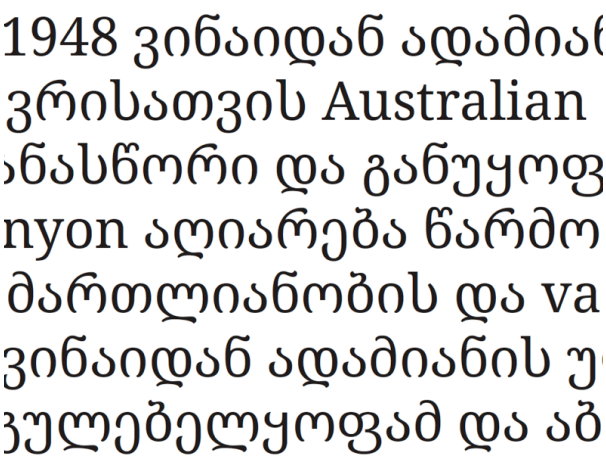


Figure 24: Droid Georgian.

Some may be wondering why I'm talking about Android at this point on the road to Noto.

Figure 25: Succession of designs.

In 2009 I was asked to adapt Droid Sans into a branding typeface for Chrome. This design became Open Sans, a slightly wider proportioned version of Droid more suitable for regular text in documents.

Roughly in parallel to this, the Chrome and Android groups at Google were discussing the idea of combining efforts on the development of a super font which would cover all of Unicode. The problem of 'tofu' displaying in Internet searches was becoming more problematic as the world's Internet usage was climbing dramatically.

With the joint packing of Chrome and Android, Open Sans then became the basis for Noto Sans and Droid Serif became Noto Serif.

By 2011 it was decided to expand on the Latin family for Noto so that it would include a full typographic palette of styles of weight and width. The sans and serif would have condensed and narrow versions added, additional weights from thin to black. The serif had an added range of contrast from super high contrast to low contrast. That's about 72 font styles per family — no longer would the polyglot typographic palette be limited by just regular and bold styles!

Figure 26: Noto, ca. 2011.

Unicode had by this time accounted for around 3400 characters to support Latin, Cyrillic, Greek and phonetic writing. This complement of characters supports over 500 languages. With these additional characters being added in all the additional weights, widths and style of Latin fonts meant drawing about 230,000 characters.

By now it was beyond clear that Noto would never ship as a single font binary with all of Unicode. Remember the old Arial Unicode font alone was 22Mb in size with just 55,000 characters. Instead, the Noto fonts are built as individual modules covering one or a few related Unicode script, or writing system, ranges.

Aside from the file size problem, writing systems beyond the Thai we drew become quite complex in form. Vertically, Thai is just barely able to squeeze into the vertical metrics of a Latin typeface. Below, you can see that Javanese (on the left) and Khmer (on the right) are even more complex vertically, and simply could not reasonably squeeze into the dimensions of the Latin script.

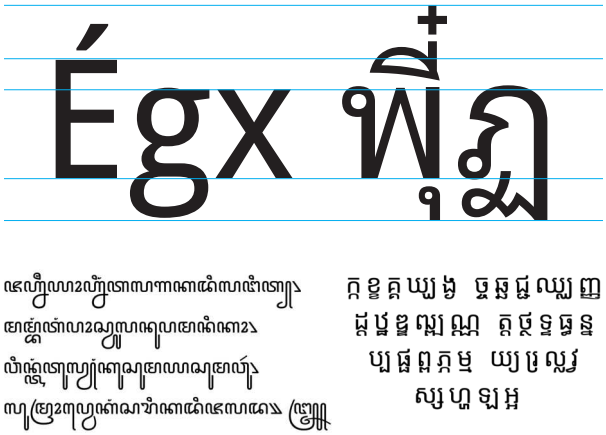


Figure 27: Beyond Latin metrics: Javanese (left), Khmer (right).

Another example is the Nastaliq style of Arabic used for Urdu, Pashto and Persian languages. While the commonly used Naskh follows a flat baseline, Nastaliq's baseline slopes downward to the left. The longer the word, the taller the dimensions become.

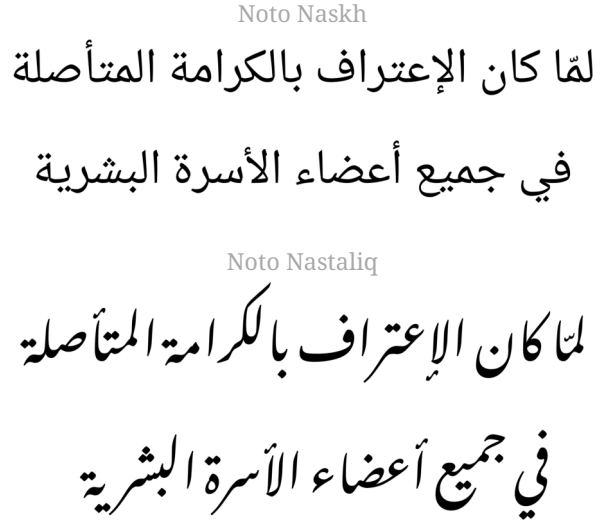


Figure 28: Noto Naskh (flat baseline) and Noto Nastaliq (sloped baseline).

One of the ways we looked at this enormous project was to break down the writing systems into categories. This way we could classify related scripts, either by their region, complexity, or relative use in the modern world.

Here, the orange boxes denote scripts that may have been dead for hundreds of years or more, while the greens are scripts used in India, and so on. This aided in prioritizing and organizing the expertise needed to complete each piece of the project.

Figure 29: Organizing scripts for Noto.

Cuneiform is a good example of the 'dead scripts' just mentioned; it may be as old as 5,000 years. While not in practical use, it is certainly useful for scholars and linguists to have encoded in a font file.

Figure 30: Cuneiform, original and Noto.

Anatolian hieroglyphs are at least 4,000 years old, thus predating Egyptian hieroglyphs. They are therefore represented as slightly more crude and smoothed over.

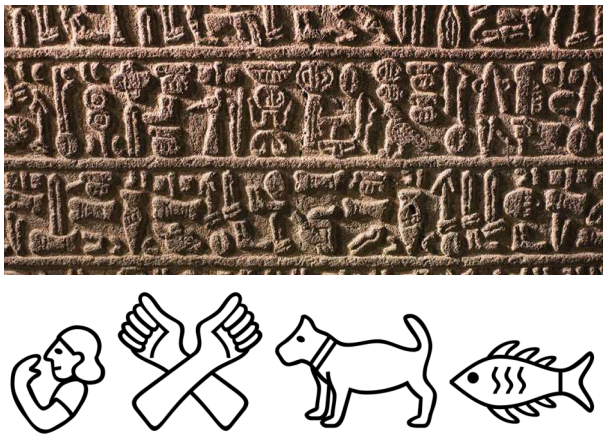


Figure 31: Anatolian hieroglyphs, original and Noto.

Egyptian hieroglyphs are much more crisp and refined in design than their Anatolian ancestors.

Figure 32: Egyptian hieroglyphs, original and Noto.

Many of the scripts are categorized as 'complex' scripts, requiring a great deal of programming to assemble words in the proper manner. Arabic, being right to left and having many forms of the same letter, falls into this class, as do the scripts used in India. Jelle Bosma is Monotype's creative lead in creating the Indic scripts, and is working on updates for Unicode version 13.

In the illustration below, the word on the right is the word 'Hindi' spelled out in Devanagari script. The top line is how it looks with plain Unicode characters set together, spelling out the word. On each successive line, you can see how the script is 're-shaped' as advanced typography tables rearrange the letters *as they are typed*. In the second line, the green characters change places. In the third line, the green and pink characters form a ligature.

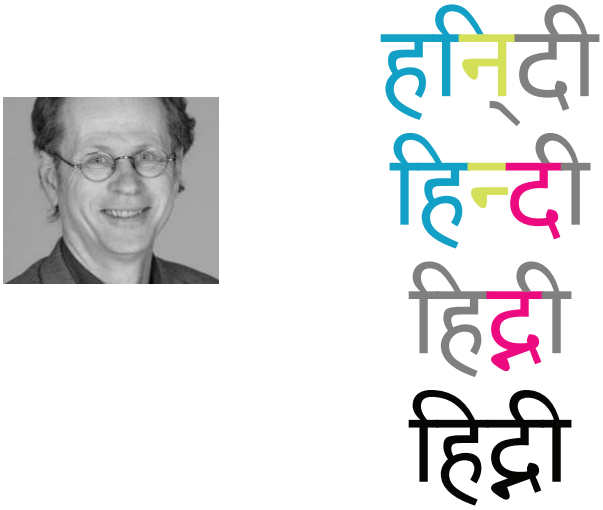


Figure 33: Designer Jelle Bosma (left); the right shows the word 'Hindi' being shaped as it is typed.

The Indic writing systems appear quite different from each other but we've designed them to harmonize as much as possible. The rectangular Devanagari contrasts quite a lot with the fluid Sinhala, but their color and proportion are preserved to keep them in sync. It's the same with Telugu and Tamil; they contrast a great deal in overall texture, but their proportions and color tie them together.

Devanagari

अनुच्छेद 1 – सभी मनुष्यों को गौरव और अधिकारों के विषय में जन्मजात स्वतन्त्रता और समानता प्राप्त हैं। उन्हें बुद्धि और अन्तरात्मा की देन प्राप्त है और परस्पर उन्हें भाईचारे के भाव से बर्ताव करना चाहिए।

Sinhala

ක්‍රි.පූ. 500දී පමණ ශ්‍රී ලංකාවට හැඳින්වුණු පැරණි බ්‍රාහ්මීය අකුරු පැවත එන බව පුරාවිද්‍යාඥයෝ පවසන නමුත් හෙළ හවුල ප්‍රධාන පිරිසක් එම මතය බැහැකරයි. සිංහල හෝඩියේ අකුරු 60ක් ඇති අතර ඉන් 4ක් මෑත කාලයේ දී ඇතුළත් වූ ඒවා වෙයි. සිංහල භාෂාවෙහි ස්වර විශාල ප්‍රමාණයක් ඇත.

Telugu

ఆంధ్ర ప్రదేశ్ మరియు తెలంగాణ రాష్ట్రాల అధికార భాష తెలుగు. భారత దేశం లో తెలుగు మాతృభాషగా మాట్లాడే 8.7 కోట్ల (2001) జనాభాతో ప్రాంతీయ భాషలలో మొదటి స్థానంలో ఉంది. ప్రపంచంలోని ప్రజలు అత్యధికముగా మాట్లాడే భాషలలో పదమూడవ స్థానములోను, భారత దేశములో హిందీ, బెంగాలీ తర్వాత మూడవ స్థానములోను నిలుస్తుంది.

Tamil

தமிழர் உலகம் முழுவதும் பரவி வாழ்ந்தாலும் ஒரு சில இடங்களில் தமிழர் செறிந்து வாழ்கின்றனர். அத்தகைய இடங்களாகப் பின்வருவன உள்ளன.

Figure 34: Indic scripts in Noto.

Just as I mentioned there were technical issues to resolve way back with Arial Unicode; the same has been true with Noto. My colleague Toshi Omagari is shown below orienting his laptop to design Mongolian, a connected script which reads top-down, left-to-right.

The tool developers for GlyphsApp have been incredibly supportive in updating their product to make these complex scripts in Noto possible. It wasn't long before they delivered a fix which allowed Toshi to see his work in a way it would be used.

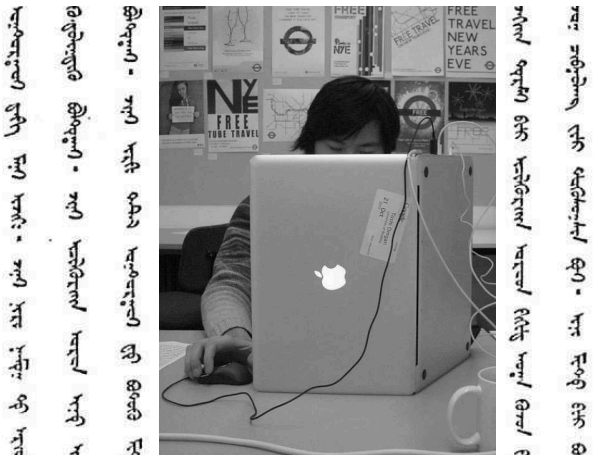


Figure 35: Toshi Omagari working on Mongolian.

Earlier I mentioned dead scripts. On the flip side is Adlam, a script developed in the late 1980s by the brothers Ibrahima and Abdoulaye Barry. This writing system transcribes the Fulani language spoken in Guinea, Nigeria and Liberia. Before Adlam, Fulani was written in either Arabic or Latin script.

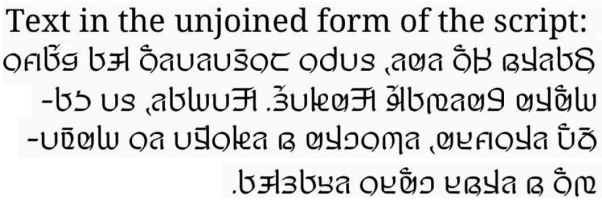


Figure 36: Adlam, developed in the 1980s.

By 2018 the Noto fonts covered nearly 64,000 characters and in the last 2 years there have been many updates. Unicode 13.0 adds 4 new scripts and 5,000 new characters to this count. "What could possibly be left?" you might ask.

Khitan, a language once spoken in Manchuria has been added, as has Chorasmian, a language of ancient Persia.

Sutton Signwriting, a notation system used to teach sign language has been added. It requires thousands of icons necessary to show hand gestures and facial expressions used by sign language interpreters.

And the Noto symbol font is getting many new characters including the long-awaited accordion and fondue dish symbols. The list keeps growing.

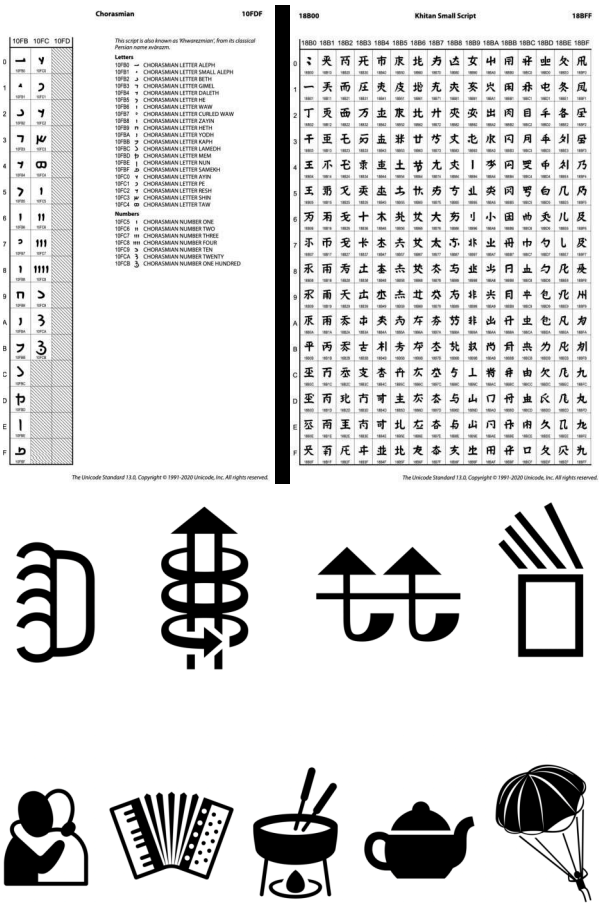


Figure 37: Added in Unicode 13.0: Khitan and Chorasmian scripts (top), Sutton Signwriting (middle), assorted symbols (bottom).

Noto will continue to be polished and refined to reflect the demands of the community of people using the fonts.

In addition to about 30 people within Monotype that have worked on the Noto project, we've been working with more than 30 outside linguists, consultants and designers around the world, notably including Fiona Ross, Tiro Typeworks, and Kigali Design.

Other people I'd like to thank here: Abdoulaye & Ibrahima Barry; Jo De Baerdemaeker; Cadson Demak Ltd; Diane Collier; Fontef Type Foundry; Kalapi Gajjar-Bordawekar; Yanone Gerner; Gajjar & Vilhjamsson Private Limited; Kimya Gandhi; Patrick Giasson; John Hudson; Indian Type Foundry; Yanek Iontef; Letterjuice Ltd.; Ben Mitchell; James Montalbano; Elena Papassissa; Rainer Erich Scheichelbauer; Zachary Scheuren; Georg Seifert; Vaibhav Singh; Anuthin Wongsunkakon; Pascal Zoghbi.

The community of users and testers who have provided feedback is, of course, much larger. It is an honor to be working for them to make this enormous undertaking a possibility.

In the end Noto may not be used for retranslating the bible or imperial decrees. It might be very simple messages that we can convey with this enormous tool we have at our disposal. And maybe Noto is another step towards that romantic notion I heard as a student — that people around the world could spend more time communicating instead of fighting.

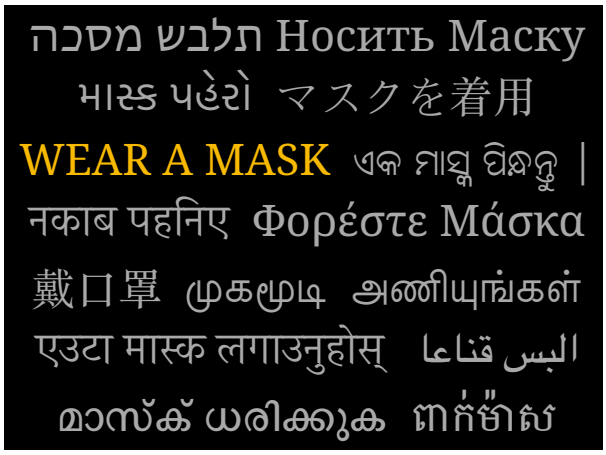


Figure 38: Translations unedited, via Google Translate.

◇ Steven Matteson
Monotype
monotype.com/studio/steve-matteson

Typographical explorations in two unicase alphabets

Jennifer Claudio

Abstract

We take for granted the convenience of expressing emotions in typography for Latin-based writing, such as using capital letters. This submission explores the use of a variety of type attributes including color, typeface, size, and distortion as they are used to convey emotional charge in Hangul (Korean) and Arabic writing.

1 Introduction

The written word has a rich history ranging from engravings through brush calligraphy and into digital typography. While typography itself is defined as "the art and technique of arranging type to make written language legible, readable, and appealing when displayed" (Wikipedia), therein exists a deeper psychological contribution to the emotional charge of words. Likewise, the developing methods and needs of written communication demand attention to typographical methods of expression, which are sometimes limited by alphabetic constraints for some language families.

The purpose of this exploration is to draw attention to typographic methods needed to nurture connections between the spoken and written language, as well as to their associated cultures, to broaden the range of expression used in more world languages.

2 Unicase alphabets

Two unicase alphabet systems will be addressed to provide specific background, Hangul and Arabic. Unicase systems do not have a differentiation of letterforms between upper- and lowercase letters, terminology that traces roots to the early typographic systems where moveable typecasts were stored in drawers with capital letters traditionally in the literal upper case.

Hangul is the writing system of the Republic of Korea, and it currently uses an alphabet constituent of fourteen consonants and ten vowels. The Hangul alphabet is described as an alphabetic syllabary, meaning that although alphabet units consist of vowels and consonants working together to depict a sound, letter and syllable combinations have both a vertical and horizontal relationship. This relationship is in contrast to a language such as English, where each alphabetic letter has only a horizontal relationship with the ones that precede or follow it.

Arabic script, comprised of twenty-eight standard letters, is used for writing several languages, including Farsi (Persian), Urdu, and Pashto, and has variations that have incorporated modifications to the syllabary such as for the Uyghur language. Although Arabic forms, including the number of recognized letters or letterforms, may vary slightly by country or culture, all Arabic script is written from right to left and has letters that change form depending on positioning within a word. Some, but not all, scripts include diacritic markers.

3 Emotion in language

Words carry only as much meaning as their context can convey. A standalone word, "what", can mean any number of things, yet when written as What, what, or WHAT — even without punctuation — it can elicit varying emotions or response from the reader. Whether the visual imagery of the word conveys surprise, doubt, even potentially anger, depends on the typographic elements of the word. Here, emotions relevant to emphasis (shouting) and endearment will be discussed in the context of typographic needs.

Shouting, or a greater volume when speaking, tends to be written with capital letters, bold face, or a size increase, and it typically occurs when a user expresses anger, assertiveness, demands, or surprise. (Tangentially, this becomes more pronounced in the realms of social media and game chat media.) While it is easy to shout using the Latin alphabet, this cannot be the case in a language such as for the Korean Hangul or Arabic. In these alphabets, besides using extra exclamation marks, shouting must then be conveyed through other typographic adjustments.

Two ways of demonstrating "louder" text that may immediately come to mind are size increases and color highlights. In comics, this is convenient, but it poses difficulty for in-line text. Although both Korean and Arabic can use italics as a usable option for emphasis, it seems that Arabic font kerning is sometimes disrupted by italicizing. Some typographic elements already serve other functions. Although characters that do not disturb the flow of the sentence can be stretched horizontally, this is not a method of changing the emotional charge of the word. A letter is more typically elongated to emphasize strokes that differentiate letters, rather than to imply a different emotional setting of the word.

Endearment or "cute" writing often incorporates letters with softer curves, and graphic designers and artists might choose to modify letters into bubbly forms or dot i's with hearts. As with the situation for capital letters, some of these modifications cannot

occur in other alphabet systems. Hangul does not have diacritics, although some of its vowels could be heart-morphed. An Arabic phrase of endearment would also not likely find itself with heart-shaped "dots", and furthermore, using baseline shifts would also be inconvenient, if possible at all, mainly due to the necessity for cursive script and due to the changes of letter shape dependent upon position.

4 Upcoming work

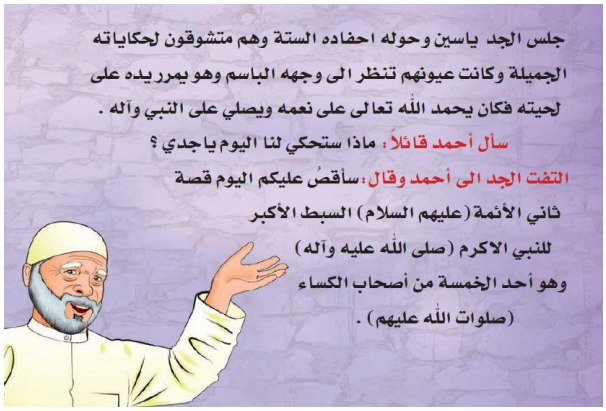
A second phase of this exploratory project will examine typographic methods and modifications in more detail, based on data relevant to perceived emotions conveyed by typographic samples in children's books, advertising media, and social media in English, Cyrillic, Hangul, Arabic, and Bangla. Future work will also address complexities and typographic modifications specific to Quranic writing.

5 Acknowledgments

Special thanks to AbdulBaqi Matrook for providing information about speaking and reading native Arabic, and for providing examples of print media that were used in the TUG 2020 presentation. I also acknowledge the creators of the AlifBee mobile app for helpful lessons on an introduction to basic Arabic and Yusuf Alam for providing early samples of Arabic calligraphy.

◇ Jennifer Claudio
San Jose, California
claudioj (at) esuhsd dot org

Example 1: Elongations are not used to convey emotion in the titles for children's books, "I'm Sorry" (left) and "Thank You" (right).



Example 2: The text in red is used to emphasize talking rather than narration.

Example 3: Boldface and type-play are used in comics for emotional emphasis.

Example 4: The Korean phrase "hangsang" meaning "always" (left), and Korean "yeonin" and Arabic "habibati" (right) for "sweetheart" (when addressed to a female), with hearts to express endearment, and without hearts, for comparison, in green.

Your personal LaTeX bookshelf: Improving your background in a time of lockdown

Peter Flynn

Abstract

This paper describes the development of a LaTeX package to create a bookshelf image from a BibTeX file, suitable for use as a background for a video call in Zoom, Skype, or similar. Each entry is typeset as the spine of a book with title and author, using a randomly-selected font, color, and size. The paper describes the problems of random choice with both fixed-length and [potentially] endless lists, and the algorithm used to fit the author and title onto the spine. The package is available as bookshelf on CTAN for inspection and testing.

Background (literally)

It started on Twitter, when several people were commenting on the way people appeared when suddenly faced with having to do a Skype or Zoom video call during the COVID-19 lockdown. Apart from the lack of a camera crew, makeup team, sound crew, and production control, there were a lot of hastily-cleared walls, bookcases, window-ledges, and even whole rooms on view behind the talking head.

In particular, people who read and write, particularly academics, have lots of bookcases with lots of books, often in a state of considerable disarray. This doesn't look good — people may laugh about notoriously untidy professors, but when you need to sit up and be interviewed about epidemiology, or seroprevalence, or the 1918 influenza pandemic, you need to look calm and professional, and that jumble of books doesn't cut it.

It suddenly dawned on me that in the BibTeX users' environment, we have title and author for practically everything we have ever cited — somewhere. What was needed was a *virtual* bookcase, an image generated from life's collection of reading.

Publishers do keep images of their books, but usually the front cover, not the spine; and even so, they would not be available to the public, nor would they ever be in a sensible, uniform location on their web sites. No, it would need to be random: a random color for background and font; a random font from the huge range available to TeX users; and a random height and width of spine. In fact the only non-random data available would be the BibTeX entries, and rather than sort them, the order could be left to the user.

'Actually it turns out to be rather easy, but it would need an algorithm for colour-pairing, and a few assorted layouts for title an author. But basically, it works.' (May 1st)

Figure 1: First pass

Start-up

In the traditional Internet ethos of 'rough consensus and running code' it didn't take too long to come up with a proof-of-concept, which I ran past `@latex_ninja`, `@damienmulley`, and a few of the usual suspects (Figure 1).

By this time the requirements were becoming more apparent:

Randomness There needed to be a way to generate random values to select at least five aspects: *a*) colors (font and background); *b*) height and width; and *c*) font (well, typeface).

Data The need for selection meant that LaTeX somehow had to be provided with a list of available typefaces and available colors, and that minima and maxima for the book spine height and width needed to be set; and that those would need to be floating-point (lengths) whereas the font and color selection would need to be integer.

Color-pairing It was clear from early on that just picking two random colors was a recipe for conflict. What was needed was a way to say if one color was sufficiently in contrast with the other one to be legible.

Format It would be nice if there was some variation in spine layout, rather than having all the books look the same.

The randomness was easily fixed with Donald Arseneau's wonderful random package, which can generate both random integers and random dimensions.

However, if this was to deal with anyone's BibTeX files, some way to deal with character encodings would be needed, some way to overcome the assorted weirdnesses of old *bibtex* `.bst` files, and some way to choose from the user's installed fonts. That most useful of devices, Occam's Razor, was employed: UTF-8 only, XeLaTeX only, using biblatex and *biber*. I've been using this method for a couple of years now, and while I'm aware of the need for more development, it works for me, and the time has probably come to put the old `.bst` system out to grass.

Implementation

A shell script was created that extracted all entry keys from the user's BibTeX file and formatted each as a command to call the `\makebook` command, which the class defines to handle one entry. This can be `\input` by the user's document.

```
cat "$BIBFILE" |\
    grep '^@' |\
    grep -viE '(@Preamble|@String)' |\
    awk -F\{ '{print $2}' |\
    awk -F, '{print "\\makebook{" $1 "}%"}' \
      >entries.tex
```

That left basically three main actions: pick a font, pick the colors, and size the spine.

Font selection LaTeX has no way to create a list of installed fonts. Operating systems provide this information, so an external preprocessor was going to be needed. A TeX `\ifcase` structure was considered, but the number of installed fonts on many systems would be too large. The method chosen was to create a set of files numbered `1.tex`, `2.tex`, etc., in a subdirectory, each one containing a font selection command. The numbering is easily scripted on Unix-like systems (including GNU/Linux and Apple macOS) by using *fc-list* and the standard text utilities to create the files, simply numbered in order of occurrence.

The final action is to place a command setting the maximum bound for the random choice into another file that gets `\input`. Selection can then be done with `\setrannum` between 1 and the maximum, and then using `\input` to execute the font selection.

Colors In the case of colors, there is again a theoretical infinity of choice. However, practicality suggested one of the named palettes in the xcolor package, and *svgnames* was chosen as a representative sample. It also had the advantage of being small enough to be instantiated as an `\ifcase` structure. Extending the script was straightforward to extract the color names from `svgnam.def` and write the `\ifcase` into a file that can be `\input`. As with font selection, `\setrannum` is used to pick a number to apply to the `\ifcase` for the background, and again for the foreground.

Height and width Random dimensions sounded fine, but needed taming: for any given length of title and author, a certain amount of space is needed. In LaTeX, this tends to be like the choice of column type in a `tabular` environment: left, right, and center only handle single lines of data: for longer data you need a paragraphic cell. So long titles need to be allowed to wrap naturally in a `\vbox`, whereas shorter ones don't, so this is going to affect how much width and height is needed. The starting-point was a height and width set with `\setrandimen` between 5–20mm wide and 70–110mm high.

In addition, an alternative layout was created: author name across the top, rather than run-in with the title. The sizing algorithm was therefore:

1. an author name shorter than the randomly-chosen width of the spine would be typeset horizontally across the width of the spine, at the top, and its height deducted from the randomly-chosen height of the spine;
2. measure the width of the typeset title (or the title and author, joined by an em dash);
3. if the result was longer than the available height, typeset the title (and author, if needed) into a box of width at the available height in ragged-right mode so that it will run naturally to as many lines as needed;
4. measure the height of that box and if necessary increase the chosen width of the spine to accommodate it.

Theoretically you could then cycle round and see those that affected the choice of where the author was typeset, but this was felt to be a step too far for an initial solution.

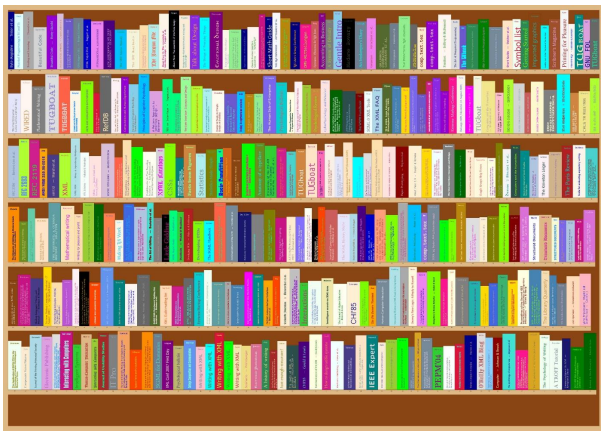
Adjustments

One immediate problem was known — colour clash or brightness and contrast in pairing — but its effect was not apparent until a large bookshelf was created. A workable solution is due to Nir Dobovizki [1], which proposes the formula

$$\text{brightness} = \sqrt{.241r^2 + .691g^2 + .068b^2}$$

where r, g, and b are the red, green, and blue values expressed as integers between 0 and 255.

Code to compute this was added to the script so that color selection and brightness selection could



'Fixed the colour-pairing and random font selection and picked two layouts. Basically working but needs more test data. Thid is my thesis bibliography' [sic] (May 16th)

Figure 2: Working solution

be done in parallel, and a clash rejected, within a loop. By inspection of the gamut, the approximate location of the median brightness value appeared to be 0.7, so the code ensures that each of the two colors chosen falls either side of this value. A notional value of 10 was used for regulating loop exit, after which the current values are used regardless; this appears to be sufficient.

This created a working solution (Figure 2), but left an unresolved issue: the data-preparation script was including all TTF and OTF fonts regardless of their type, whereas it needed limiting to text typefaces with a Latin register (that is, excluding math, symbols, and display fonts). In addition, on the author's system, some directories of older, experimental, and test fonts needed to be excluded.

Some inspection and experimentation showed that a reasonable list could be created by excluding any font name with a match in a regular expression containing suitable strings:

```
(Bitmap|Emoji|Dingbats|Jazz|STIX|dings|
 Symbol|Numeric|DIN|Ornament|OCR|CJK|
 Awesome|Dummy|Math)
```

A cyclical pattern of test-as-you-go had been established, and I am grateful to the numerous people who sent me their thesis BibTeX files. One late addition was to shade the background to a dark color for the inside of the bookshelf, and to color the shelves themselves a pale cream, for which I used a technique suggested by Ulrike Fischer [2].

The final stage, left to the user, is to convert the PDF to image format. The default size is a landscape A0 page, which is huge, but accommodates a few hundred volumes. It shrinks well to a screen size.

Conclusions

The most recent step was to put the package on CTAN and see if there were suggestions (none so far). By this time a number of helpful suggestions had been received, and offers of testing were accepted. By May 24 I was able to report on Twitter:

> May 24 ▪ Replying to `@latex_ninja` `@TeX4Publication` `@erdmaennchen42` It has just been uploaded to CTAN. Thank you.

What could be done better?

- The script works in *bash* (Linux) and *zsh* (Mac). It needs extending to Windows (*cygwin*? *Powershell*?);
- The colors currently are too bright on-screen, although reportedly OK for printing: perhaps the color selection algorithm needs revising;
- Some more spine layouts would be interesting, as would more bookshelf layouts: books at an angle, or stacked horizontally;
- Can something from the biblatex field selection provide for a place to store color, font, layout, and size as one would for bibliometrics or a *catalogue raisonné*;
- 180° rotation for spine titles is needed for some non-English languages, and math in titles needs more testing;
- In essence, this is just an output format from a `.bbl` file. Perhaps it would be more useful rewritten as a biblatex style option.

References

[1] N. Dobovizki. Calculating the Perceived Brightness of a Color. *Making Time-Tracking Software*, Apr 2008. `https://www.nbdtech.com/Blog/archive/2008/04/27/Calculating-the-Perceived-Brightness-of-a-Color.aspx`

[2] U. Fischer. How to set a certain color (other than white) to margin areas? *tex.stackexchange.com*, Dec 2010. `https://tex.stackexchange.com/questions/7725/how-to-set-a-certain-color-other-than-white-to-margin-areas`

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

Noticing history — a personal view*

David Walden

Abstract

I presume most of us who participated in or watched the TUG 2020 conference are not professional historians but rather are computing practitioners or users of computing technology such as TEX and friends. We have access to memories, papers, and flexibility in what we study, and how we present what we learn, that won't all be available to professional historians. I believe it is our job to help the computing history world capture more computing history while it still exists to be captured.

Introduction

Although I am speaking today about how computing history is done, I am not a formally trained historian. I come to the views expressed in the rest of this paper from a thirty year career in the technology and business of computing and now 25 years of amateur research and writing about computing history. During this time I have been a near-constant user of computing technology — since retirement from business, including LATEX and other components of the TEX-based-or-derived infrastructure.

Over the past 15 or 20 years of my involvement in researching computing history, I have become acquainted with a number of professional, often academic, historians and have learned something about what they typically do. I have come to believe that the professional historian can't do it alone, and history work might use some help from the likes of us.

For much of this paper I will talk about computing history and the people who work or have worked in computing. Everything I say is equally applicable to typography, typesetting, and printing history and the people who work or have worked in those activities. In this paper I will refer to "we" or "us"; by this I will mean people such as are at this conference and who are typical readers of *TUGboat* — people who are involved in the development of computing, typography, typesetting, and printing technology or closely observe or seriously use it.

A recurring theme of the rest of this article is the distinction and separation of amateur historians who themselves actually experienced, participated in, and understood the formations and transformations of various computer-related fields, from the professional historians who report, translate, and interpret the histories of those fields.[1]

I don't mean to be critical of professional computing historians. They have had added essential scholarship, stability, and validation to the field of computing history. While collecting and writing computing history initially was pioneered by computer people themselves who were afraid the history of their field was being lost, it would not have become the vibrant and distinct field it now is had it not become a branch of academic history. I do mean to encourage people from the computing field to engage more in capturing and recording computing history in ways that complement and supplement the work of the professionals and may be valuable in their own right. I also hope to suggest to the professionals how valuable, even essential, our amateur history efforts can be.

In the rest of this paper I will discuss three topics.

1. History is moving fast, and many memories and materials from history are being lost; official historians can't capture and document enough computing history by themselves; and thus the history world needs our help.
2. We have useful skills and abilities to contribute.
3. There are many ways in which we can help.

1 The need

Things have been changing fast in the decades since the 1940s and 1950s.

After centuries of using essentially Gutenberg technology and about 70 years of dependence on Linotype, Monotype, and other forms of mechanical composition, the history of phototypesetting zipped by in a couple of decades, and desktop publishing went from infancy to ubiquity in another couple of decades.

Several generations of people who participated in developments in those years have died or are getting old as are people who closely observed the early developments. While many important pioneers have been interviewed and have sent their papers to archives such as the Charles Babbage Institute (CBI), the Computer History Museum (CHM) or their university or corporate archive, there are other major pioneers and *many lesser pioneers* whose histories need to be captured — now. (As history continues to run along, there will always be more people and projects whose histories should be captured.)

We also need the documents of computing developments: hardware diagrams, program listings,

*This paper is derived from a presentation at the TEX Users Group's 2020 annual conference (carried out via Zoom). The slides that went with this presentation are at `tug.org/tug2020/preprints/Walden-history-slides.pdf`. This written version of the presentation does not closely follow the slides.

project plans, company plans, and so on. In some cases such materials have been archived. For example, lots of materials from the Control Data Corporation are in the Charles Babbage Institute archive (`www.cbi.umn.edu`). Some of the Aldus (PageMaker) company's annual reports and other documents are in the archive of the Computer History Museum (`computerhistory.org`). Perhaps more typically, few of the materials of Interleaf Inc., a rival of Aldus in the 1980s, are in a formal archive. Some of the archiving may come from a person possessing material volunteering it to an archive; in perhaps fewer cases, an archive recruits materials from a company or individual. Whatever the case, there is much more that should be collected for which there may be no current plans for collection.

In this digital age, it is more important than ever to capture such materials now. Before, when everything was on paper, there was at least a chance that the material would eventually be able to be collected. Today companies' and individuals' computers get discarded with no one thinking about what they may contain of historical significance that was never on paper.

Professional computing historians cannot possibly do all the desirable history work.

The professionals of course do lots of collection, research into, and publishing of history, including interviewing people from the historian's historical era of interest. But there are many more people who might be interviewed or who might be encouraged to write their memoirs.

Computing historians tend to work on history that is some number of years in the past. I suppose that it is not history unless it is sufficiently in the past. Thus professional historians often won't be involved in what's happening now and won't be collecting it as it happens.

Professional historians also often write for a specialized audience. Computing history for the masses tends to be the domain of authors of books and articles working in a more journalistic style, for instance books such as *The Soul of a New Machine* by Tracy Kidder; *Where Wizards Stay up Late* by Katie Hafner and Matthew Lyon; *The Dream Machine* by M. Mitchell Waldrop; and *The Innovators* by Walter Isaacson. The professionals sometimes are dismissive of such journalistic writing when it comes out; and later, naturally, the historians will consider such writing less useful than primary sources, however contemporaneously written or thoroughly researched a book or article may have been.

There are other problems that lead to computing history not being collected by the professionals.

To some extent the traditional academic history world looks upon computing history as belonging in some other academic department, and computing historians sometimes have had a hard time getting jobs in those academic history departments. They sometimes are in informatics departments or maybe library departments.[2]

Even computer science departments, which one would think should be interested in computing history, are not interested enough to spend a faculty position on a professional historian of computing. Some more-or-less history books come out of computer science departments, for instance *The Multics System* by Elliott Organick and *The Origins of Digital Computers — Selected Papers* edited by Brian Randell, but these tend to come from computer people rather than from official historians.

There is also a lot of history that the professionals tend not to focus on. They are less likely to do research aimed at writing straightforward accounts about what happened with a project or technology — what many of us may think of as the usual way technology history is written. Historians mostly are more interested in the political, social, etc., context of a technology development rather than in the details of the technology.[3] They also tend to work on that for which they can get grant funding and which is done in a way which gains them the respect of other professional historians and eventually tenure at their academic institutions.

There are business forces that lead to computing history being lost.

When new management comes into a company, it may discard lots of historic material as part of its push to clean up the company (Figure 1). Or the new management may not care at all about historical value. I can hear a new owner in the business of asset stripping saying, "We are in this to sell off the company's assets. A bunch of long ago published technical reports done on government contracts and therefore in the public domain aren't worth anything. We are not in the business of saving stuff for its intangible historical value. Shred it." Also, when a project ends, a company is sold, or a when a company goes out of business, the company's materials are often discarded — I have seen this in person; probably you have too. The use of off-site storage by companies as a way to *hold onto* materials despite limited storage space can be another problem. Boxes of documents that go to off-site storage sometimes are never found again — this also has happened to me.

Figure 1: Sights we have seen all too often.

After my talk, Chuck Bigelow noted to me that much of what I am saying is not new. In the history of printing the older information technology was lost.

> In the first centuries of printing, most typographic materials other than books were lost. Books were preserved as valuable information containers accessible to general readers, but book-making tools understood only by a few specialists were rarely preserved. Printers went out of business, type wore out and was melted down to make newer type, presses wore out and were replaced. Technical know-how kept as trade secrets was often lost when the keepers of the secrets passed away. Early type designers, typographers, and printers didn't write about the details of their work. It wasn't until near the end of the 16th century that printing types, matrices, molds, presses, and account books began to be preserved with greater frequency. Historians now pore over ancient records and surviving materials, trying to extract facts from indirect evidence.[4]

The professional historians can't cover history alone. They are not in a position to gather primary source material as it happens; there are materials they may never learn about; and there are aspects of history work they are not motivated to do. The history world needs our help. In his book *History Hunting* (I will say more about it later), James Cortada notes that writing real history always means going to primary documents. People like us are well positioned to collect primary documents. Now is the time to collect them.

2 Our special qualifications

Computing, typography, typesetting, and printing technologists and technology developers and users (like us here at this conference) have some special qualifications that can let us supplement what the professionals are able to cover.

First, there are lots of us and we are in lots of places. Also, more of us are being trained or otherwise going into our fields all the time — in much greater numbers than historians are being trained. There are few of them and they tend to be in academic institutions rather than in the locations where the history is happening.

Second, we are or have been part of or close observers of history unfolding. Today's historian may research and interpret how, for instance, 18-bit minicomputers were used in the 1960s or the impact of early digital typesetting systems. Some of us used those systems, which gives us a different, perhaps complementary, perspective on the history of the technologies.

Third, we have knowledge, skills, or resources the professional computing historians may not have (just as they have skills we do not have). We can write computer programs. We can read computer program and circuit diagrams. Some of us have led or been part of big computer-based projects in business or may be leading or part of significant open source projects, perhaps giving us deeper perspectives on how technology is developed and more able to apply the power of teamwork to get bigger things done (my feeling is that academic historians tend to work more individually or in smaller teams). If retired, we may have time that the professional historians do not have. Some of us may even have money we can contribute to history work or institutions.

Examples of computing history projects that technologists have accomplished, and historians probably would not have, are Zbigniew Stachniak's project to recover what was on Micro Computer Machines cassette tapes[5] and Len Shustek's project for recovering what was on the Computer History Museum's large collection of vintage magnetic tapes.[6]

One might argue that the historians know how to do a lot of things we do not know how to do. But maybe we can learn, for instance from books such James Cortada's *History Hunting — A Guide to Fellow Adventurers* (Figure 2).[7] Cortada had a long successful career at IBM after he was trained in college as a PhD historian. In the later years of his IBM career and since he retired from business, he has done a vast amount of writing about the history of the computer business. His *History Hunting* book provides encouragement and guidance appropriate to other people who have become amateur historians after a technology or technology business career.

More specifically, we can learn to do oral history interviewing. It's one way to ease into collecting history. There are lots of us who have been part

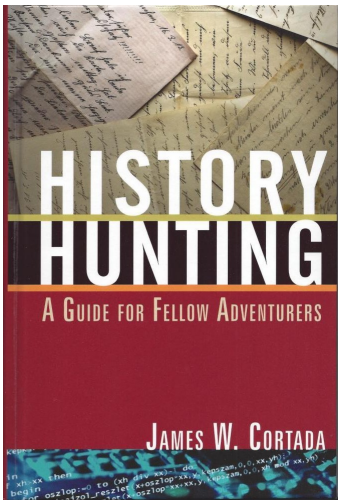


Figure 2: *History Hunting* by James Cortada. This book was highly inspiring to me and first got me thinking seriously about how we can contribute usefully to collecting and distributing computing history.

of interesting computing, typography, typesetting, printing, or publishing history and we can interview each other. Donald Ritchie's book *Doing Oral History — A Practical Guide* is a good reference as one starts doing oral history interviewing.[8,9] (At least in the TEX/TUG world, there are ready places for publishing an interview one has done: our Interview Corner, `tug.org/interviews`, or *TUGboat* and journals of other TEX user groups.)

We can also read history by the professionals and learn about doing history work from that. For instance, take a look at history written by Thomas Haigh (`tomandmaria.com/Tom/AboutMe`); he writes history that is highly scholarly and also lots of fun to read. Take a look at his paper on the history of word processing[10] or the book he co-authored on the history of the ENIAC computer.[11]

Ideally one might hope to collaborate with a professional historian as a way of gaining new skills, but that typically is not the way history is done even when both collaborators are professional historians.[12] More generally, some professional historians seem dismissive of amateur efforts even though we lived through it and we may be as academically qualified in our fields as they are in theirs. They may value hearing about what we saw but not our ability to properly interpret the history.

Historians at museums and archives such as the Computer History Museum, Charles Babbage Institute, the MIT Museum, or MIT Archive (to name just a few) may be more interested in the contributions of amateurs as well as providing service to amateurs. Naturally they may want some of our papers or artifacts. They may also give guidance about collecting history. When a computer person wants to pass historical materials to the museum or archive, the person may be put in contact with an archivist who will be receptive but also enforce the organization's policies regarding ownership, copyright, and so on.

We also can help each other learn. In particular some of us are slightly into the official computing history world, as a history journal editorial board member or leader of a small museum such as the Vintage Computer Festival Museum (`vcfed.org/wp/vcf-museum`) or doing a history project which put a person in touch with the professionals and thus better able to provide pointers for getting involved with collecting and recording history.

3 What we can do

We can create history content.

For instance, we can interview people, even lesser contributors to computing history or the narrower history of the TEX world. We can and should write our memories down rather than just telling them to each other on discussion lists. Much interesting history is exchanged, for instance, on the Internet History list (`elists.isoc.org/mailman/listinfo/internet-history`), but it is not being processed to make it better organized or more accessible beyond the raw messages being exchanged. I believe there are many interesting volunteer or academic projects that could be developed out of the ih archive. *TUGboat*'s editors seem receptive to history-oriented papers — I can point you to examples. The *IEEE Annals of the History of Computing* is all about history and always looking for submissions; quite a few *Annals* publications have been on topics close to what TUG is about. Some of those have been peer reviewed publications (including from practitioners of computing rather than historians), but the *Annals* also has a department for non-academic submissions — the Anecdotes Department (`annals-extras.org/anecdotes`); I will be happy to talk to anyone about a possible anecdote submission to the *Annals*. Finally, non-academic historians among us who are serious enough and research deeply enough can write papers that are just as scholarly as those by professional historians, albeit perhaps a somewhat different kind of history writing. Charles Bigelow, who is well known to the TEX community, has provided a recent example of scholarly writing with his history of digital fonts.[13] If writing is hard, we instead can record memories with digital audio or digital video.

We can create a website and post things there. For instance, take a look at Tom Van Vleck's wonderful `multicians.org` website. Spend some time looking around it if you don't know it already. It includes discussion about how to maintain a history website. If you do create a website, think about inheritance planning for your website and the valuable history material you collect. Another example of creating useful history material is Nelson Beebe's massive database of bibliographic information (`math.utah.edu/~beebe/bibliographies.html`). A third example is Luc Devroye's encyclopedic website compilation on typography (`luc.devroye.org/fonts.html`). If a website is too big a job, we can at least create web pages and find somewhere to post them. For instance, a web archive of material uncovered in writing a history of (the previously-mentioned) Interleaf is at `annals-extras.org/dtp/interleaf`. Another example is Paul McJones's web page of research into the history of Fortran (`softwarepreservation.org/projects/FORTRAN`). We can make unpublished or public domain materials we uncover in writing history easily available to the next researcher — not just cite existence of the materials.

We can give presentations. The Vintage Computer Festivals are annual conferences with interesting presentations. Someone has to give those presentations. It could be you. Of course, TUG also has an annual conference at which history presentations could be made, and there are half a dozen other annual TEX user group conferences that may be seeking presentations.[14]

An example I recently became aware of technologists collecting and publishing history is the Tampere International Center for Signal Processing (`annals-extras.org/pubs/TICSP.pdf`) where they have collected copious histories in various areas related to signal processing and published what they have collected. Radomir Stanković writes:

> We were guided by the general idea that looking into past helps to determine roads to the future. We believe this also is correct in the more specific case of technology — knowing the ways of thinking of scholars in the past might help reveal new ideas or avoid unfruitful approaches. We believe that a researcher needs to know the work and activities of current colleagues — equally important, know the work of "previous" colleagues — to know to some level of depth the history of the field.

We can save and/or organize historical content to which we have access, either formally or informally.

Save your papers, and find a place to send them or at least scan them and offer the scan to a stable archive.[15] Post scans of your papers and stories you write on the web in an organized way; then it at least goes to the Internet Archive (you can tell them to do a pass over your stuff).

Grab stuff when a project or system is being shut down. Roger Roach, the last CTSS system administrator, captured all of the CTSS documentation and digitized it when MIT's CTSS system was shut down in 1973. Grab stuff being thrown out. Jake Feinler was with the Network Information Center at the Stanford Research Institute. When the activity was shut down, she took all the documentation the NIC had collected over the years home to her garage. Eventually she was able to give this extensive and valuable part of Internet history to the Computer History Museum where she spent time organizing the material for the user of future researchers.[16]

Gather material that other people contribute; for instance, become the website maintainer for an organization so you can organize the organization's materials and make sure it has some place to go in the long term. Gathering other people's materials may seem like second class work, but I think I have heard at least three great people, Daniel Boorstin, Stephen Jay Gould, and E.O. Wilson, make the observation that aggregation and taxonomy can be just as valuable as original work.

Place the history you have or can capture in some stable location. David Brock of CHM and Jeff Yost of CBI assure me that they accept paper documents, scans of paper documents, and documents born digital. CHM also accepts non-paper artifacts.

Whatever you collect, organizing it to make it more accessible is important. Maybe you can take the time to create a finding guide, or organize it so it is searchable in a more sophisticated way than a Google-type search.

Regarding documents that are created as part of an effort on which you are working, numbering them sequentially with a listing of all their titles, authors, and dates that gets updated every time a new document is written improves the odds of the material being saved. The RFC list is an example of this. Had they not been numbered and rather just been a lot of documents, I doubt they would have been as successful and long lived as they are. While the documents may never be important enough to collect, if the project turns out to have been important its numbered documents have a bigger change of still existing.

We can publish historical content more or less formally. I already mentioned submitting papers to *TUGboat* or the *Annals*. We also can self-publish monographs; one example is the "commemorative brochure" written for the 50th anniversary of the CTSS system (`tug.org/l/walden-ctss`).

Another possibility is to post your memories at the Engineering, Technology, and History Wiki (`ethw.org/Main_Page`). This history resource is sponsored by half a dozen or so professional societies and is managed by the IEEE History Center. They want technologists to contribute to the site. Go look at it.

The TEX community is world wide. Among us, we are in a good position to report histories from each of our counties. More of this would be highly interesting.

We can also self-publish our memoirs; for example see the memoir of Severo Ornstein, *Computing in the Middle Ages — A View from the Trenches, 1955–1983*, which is posted at the Computer History Museum.[17] The Engineering, Technology, and History Wiki mentioned just above is very welcoming of memories of people who come from the work the wiki covers. I posted a partial memoir there. You can too.

We can join relevant organizations and do what we can to help.

We can subscribe to journals such as the *Annals*, contribute, and perhaps one day be appointed to the editorial board (this happened for me[18]). SIGCIS (`sigcis.org`) (nominally a part of the Society for the History of Technology[19]) is where professional and amateur computing historians from all over the world communicate with each other about their projects, post calls for papers, discuss book releases, and so on. If you are researching a history topic and wanted to know where to find something or how other people view what you are thinking, SIGCIS is the place to ask your question. Britain's Computer Conservation Society (`computerconservationsociety.org`) is a good organization to know about, and it has an excellent free journal called Resurrections. It is loaded with examples of stories from history that demonstrate that any of us can write up a bit of history.

One can become a docent, or join a committee,[20] or help with a project at someplace like the Computer History Museum.[21] A few years ago, my friend Guy Fedorkow began thinking about what he would do after he retired from his position as a computer system architect in a router company. Work took him between home in Boston where he lives and Silicon Valley many times each year. He introduced himself to the curators of the Computer History Museum and did a volunteer project with the IBM 1401 restoration team.[22] Having understood Guy's capabilities, the CHM curators introduced him to curators at the MIT Museum. From there, a project evolved to combine the many Whirlwind computer paper and magnetic tapes in the CHM archive with MIT's deep collection of Whirlwind project reports and notes in order to learn more about the history of software on the machine.[23,24,25]

Guy, with help of many others at both MIT and CHM, has figured out how to read the old magnetic tapes,[6] has written a Whirlwind simulator, and now is writing a paper about the effort which he is submitting for publication to the *IEEE Annals of the History of Computing*. This is another project that most professional historians probably would not have undertaken.

Let me make explicit the underlying theme of my presentation at TUG 2020 and this paper derived from it. *Much can be done to capture and publish history (e.g., of computing generally or TEX/TUG-related topics more specifically) if someone wants to do it.* It is no different than any of the projects we heard about at TUG 2020. Someone got interested in a topic, eventually put lots of time into it, perhaps recruited some help, and got something big done, e.g., Pandoc or LATEX 2ε and its successors.

Another good examples of this comes from the activities of Luanne Johnson and Burt Grad (Figure 3). They saw a need a few decades ago to save the histories of companies in various software business. Companies came and went and their histories were being lost. They started working on it. They organized meetings of pioneers in various software business areas. They interviewed them. They transcribed meeting discussions. They got software business pioneers to write papers for the *Annals of the History of Computing*. Of course, they didn't do it alone; over time they developed a little organization. There is a website at `annals-extras.org/pubs/2020-06-22-lij-sisig-website.pdf` that summarizes their activities over the years.[26] Now, as they grow older, they have arranged for the Computer History Museum to take over their archive and some of their work. Being from the computing industry, they saw the need and they did something about it which eventually became a major computing history research resource. Their most recent effort (2017–2020) was in a TEX-related area — the history of desktop publishing (`annals-extras.org/dtp`).

Figure 3: Len Shustek, Chairman of the Computer History Museum, presenting an Achievement award to Burt Grad (remotely in the Beam robot) and Luanne Johnson in March 2017 — photo credit to © Douglas Fairbairn Photography. (Photo used with permission of the individuals in the picture.)

The third person in the image is Len Shustek, who founded the Computer History Museum — another example of an individual who is not a professional historian but who has had a major effect on the world of computing history (the CHM founding is described at `tug.org/l/shustek-museum`).

In the typography and printing field, Frank Romano started his career at Linotype, continued it in phototypesetting, and now is collecting history — now of digital typesetting.[27] Most of us will not do as much history work as Frank; he can be an inspiration to us to do what we can.

The entire TEX infrastructure (CTAN, TEX Live, LATEX, *TUGboat*, conferences on all manner or topics, and so on) is an example of one or a few people deciding to do something, doing it, being joined by other people, and the result being an important contribution to the world. Maybe there could be a bit more explicit infrastructure for collecting history. More specifically, perhaps it could be an explicit goal to have more history articles in *TUGboat* and more history presentations, panels, and sessions at TUG conferences.

People like us can make contributions to capturing, organizing, and publicizing computing history or the history of our special area of interest. These contributions may be big or small.

No one can know what aspects of computing history and how it has been collected and interpreted will be important in the future. Ultimately it may not matter who managed to save the historical record. As the Bigelow quote on page 162 suggests, it just matters that somehow the history gets passed along from the people who "experienced, participated in, and understood the technological formations and transformations".[28] I claim that we practitioners and users are in as good a position as anyone to decide what should be saved and may be in the best position to contribute to the passing along. Systematically collecting, organizing, somehow archiving, and writing about what seems important to us is (1) better than indiscriminate collection of everything (for instance the Library of Congress's effort through 2017 to collect every public Twitter tweet), and (2) better than collecting nothing because it's the job of someone else.

Acknowledgments

I have learned what I know about doing history from a succession of editors-in-chief, associate editors, and editorial board members of the *IEEE Annals of the History of Computing* and from the activities in which they involved me. I have especially benefitted from near-constant collaboration since 2014 with current associate editor-in-chief David Hemmendinger. My connections through the *Annals* have led to interaction and learning from many other historians of computing.

Paulo Ney de Souza helped me prepare my TUG 2020 presentation for online showing; I am sure there are others on the program committee that I should be thanking.

I greatly appreciate information, insights, and corrections as I created the presentation and drafted this paper from Barbara Beeton, Karl Berry, Chuck Bigelow, David Brock, David Hemmendinger, Kris Holmes, Alex Magoun, and Jeff Yost.

Notes

[1] Paraphrasing slightly an observation made by Charles Bigelow, email of 2020-08-01.

[2] David Hemmendinger has reminded me that university history departments being fussy about what constitutes "real" history goes beyond computing history. History of science and history of technology scholars sometimes have been relegated to a department different than the main history department. Alex Magoun noted further to me that History of Science departments initially did not want technology history tainting their departments. It's human nature, I suppose. Each established discipline is unwelcoming to new branches of the discipline.

[3] Something about how writing computing history has evolved may be found in Martin Campbell-Kelly's paper The History of the History of Software (*IEEE Annals of the History of Computing*, vol. 29, no. 4, 2007, pp. 40–51). In it he notes that over time writing about software history moved from writing primarily about

technology to increasingly writing about what he called "supply-side industry", applications, or institutional, social, political aspects of software. Donald Knuth took exception to Campbell-Kelly's view in public lectures in 2009 and 2014, preferring the more traditional technology focused approach and worried about the "dumbing down" of computing history writing. The 2014 lecture (youtube.com/watch?v=gAXdDEQveKw) in which Knuth said the changed direction of history caused him to cry also caused a big stir among computing historians and was extensively discussed and denigrated in discussions at sigcis.org. Campbell-Kelly explained in a follow-up comment (Knuth and the Spectrum of History, *IEEE Annals of the History of Computing*, vol. 36, no. 3, 2014, p. 96) that he felt he was halfway between what Knuth wanted and what people with a more social science perspective want. Thomas Haigh wrote a follow-up article on the debate: The Tears of Donald Knuth, *Communications of the ACM*, vol. 58, no. 1, 2015, pp. 44–44, tomandmaria.com/Tom/Writing/CACMKnuthTears.pdf.

[4] Email of 2020-07-24.

[5] Software Recovery and Beyond, *IEEE Annals of the History of Computing*, vol. 41, no. 4, 2019, pp. 110–118.

[6] Magnetic Tape Data Recovery, Vintage Computer Festival West 2020, August 1, 2020, streamed via YouTube, available at youtube.com/watch?v=sKvwjYwvN2U.

[7] Published by M. E. Sharpe in 2012; a review is at tug.org/l/cortada-review.

[8] Oxford University Press, paperback edition 2014.

[9] There is also various online instruction in interviewing, for instance from the IEEE History Center: ieee.org/about/history-center.

[10] Remembering the Office of the Future: The Origins of Word Processing and Office Automation, *IEEE Annals of the History of Computing*, vol. 28 no. 4.

[11] Thomas Haigh, Mark Priestley and Crispin Rope, *ENIAC in Action — Making and Remaking the Modern Computer*, MIT Press, 2016.

[12] Thomas Haigh, mentioned in the prior paragraph, has been increasingly working in the non-traditional collaborative way, and his co-author Mark Priestly comes from the practitioner rather than professional historian world. Maybe it's the beginning of a trend.

[13] Charles Bigelow, The Font Wars, parts 1 and 2, *IEEE Annals of the History of Computing* vol. 42, no. 1, 2020, pp. 7–40, computer.org/csdl/magazine/an/2020/01.

[14] Presentations at our various TEX-related conferences (e.g., for ConTEXt, DANTE e.V., etc.) may be more varied and thus easier to become a speaker at than at conferences of more specialized groups such as the American Printing History Association or the Society of Typographic Aficionados. Our TEX-related journals may also be easier to get published in than more specialized journals. TEX of course is a fairly specialized topic, but we appear to use it as a jumping off point to whatever we want to talk or write about.

[15] Lots of famous people, knowing their papers are important, have sent their papers to archives. Less famous people can also contribute. We can too. My friend Alex McKenzie was involved in the early days of the Internet — not at the level of impact of Bob Kahn or Vint Cerf — but still his papers get plenty of use at the Charles Babbage Institute (archives.lib.umn.edu/repositories/3/resources/242)

[16] Elizabeth Feinler, The Network Information Center and its Archives, *IEEE Annals of the History of Computing*, vol. 32, no. 3, 2010.

[17] Severo Ornstein, Computing in the Middle Ages, computerhistory.org/collections/catalog/102785079. Ornstein started computing on MIT's pioneering Whirlwind computer, was part of the team that developed the LINC computer, was an Internet pioneer, was involved in developing one of the earliest music transcription programs (Mockingbird, computerhistory.org/blog/rediscovering-mockingbird-a-composers-amanuensis/), and was part of the Alto development team at Xerox PARC.

[18] walden-family.com/ieee/my-history.html

[19] The Society for the History of Technology itself has an interesting history with respect to what this paper is about: vqronline.org/essay/technology-history-and-culture-appreciation-melvin-kranzberg

[20] See for instance softwarepreservation.org.

[21] As I sat in the virtual TUG 2020 conference, I wondered, "How is what a docent does going to change in this days of COVID-19 and Zoom-based communication. Will docents only be leading virtual tours of museums?"

[22] Guy Fedorkow, About the Computer History Museum's IBM 1401 Machines, tug.org/l/fedorkow-1401; IBM 1401, A Modern Theory of Operation, ibm-1401.info/IBM-1401-Theory-of-Operation-GF.pdf

[23] Guy Fedorkow, The Whirlwind Computer at CHM, computerhistory.org/blog/the-whirlwind-computer-at-chm

[24] Guy Fedorkow and David Brock, Jingle Bits: Auditory Maintenance, Whirlwind Holiday Songs & the Dawn of Computer Music, tug.org/l/fedorkow-jingle

[25] Guy Fedorkow, Gambling on Whirlwind: How the US Navy Spent $3 Million+ and Got a Computer Game, tug.org/l/fedorkow-whirlwind-gambling

[26] Luanne and Burt also recently had descriptions of their work published in the third 2020 issue of the *IEEE of the Annals of Computing*: Preserving the History of the Software History by Luanne Johnson; In Search of Software History by Burt Grad; and Finding Software Industry History also by Burt Grad.

[27] tug.org/TUGboat/tb36-1/tb112reviews-romano.pdf and tug.org/TUGboat/tb41-1/tb127reviews-romano.pdf

[28] Charles Bigelow email of 2020-07-31.

◇ David Walden
walden-family.com/texland

TEX in church: A typographical adventure

Paulo Roberto Massa Cereda

Abstract

This article presents the author's typographical adventure when producing material for masses and church-related activities.

1 Introduction

As a result of the Second Vatican Council, the Constitution on the Sacred Liturgy, promulgated by Pope Paul VI, in articles 28 and 30, states that, to promote active participation, the people should be encouraged to take part by means of acclamations, responses, psalmody, antiphons, and songs.

The author chose two topics, which are related to each other, to cover in this article, namely songsheets and song booklets, describing how TEX and friends are used in producing them. It is worth mentioning that there is yet plenty of room to cover in other areas of the church as well. Keep in mind that there is more to the subject than meets the eye.

2 Songsheets

Sheet music is a handwritten or printed form of musical notation that uses musical symbols to indicate the pitches, rhythms or chords of a song or instrumental musical piece. The term *score* is a common alternative and more generic term for sheet music. The author uses the term *songsheet* which typically refers to a document containing the lyrics of a song alongside its musical representation.

So, a liturgical songsheet is a particular instance of the broad, general sheet music in a religious context, right? In a manner of speaking, yes, but there is something very important to be addressed. In this case, text and melody cannot be dissociated as in secular music. Some scholars consider the text to be food for the mind and melody to be food for the heart. Hence, both contribute to the complete fulfillment of a person's relationship with God. So a liturgical songsheet is not a mere score, it offers a special bond between human and divine and thus its typesetting deserves dedication to the best of its typographer's heart and soul.

Since the 1960s, permission has been granted to celebrate the Mass in vernacular languages, as seen in the Constitution on the Sacred Liturgy from the Second Vatican Council. It has been emphasized that the language used should be known to the gathered people. In the author's parish, only a very reduced number of hymns still remain their original Latin form, such as *Tantum ergo*, *Panis angelicus*, *Salve Regina* and *Anima Christi*. Several other hymns were composed or translated using the vernacular language, which is, in the author's case, Brazilian Portuguese.

However, in small communities, the liturgical music tradition was devotedly kept through verbal heritage. There were no songsheets available or musically knowledgeable people to interpret them. The author's community had to ensure that newer generations learn songs by constant repetition and usage as a means to perpetuate them. This was basically the continuity of hermeneutics for future generations employed since the 1960s to the present day.

The community had nothing documented. The author being a music hobbyist, he decided to take a step further and transcribe as many liturgical songs as he possibly could. Transcription, for those unfamiliar with the term, is the practice of notating a piece which was previously unnotated (that is, not recorded in musical notation) as written music. The author takes some freedom regarding the pitch and, of course, due to the nature of verbal heritage, some melodic inaccuracies might arise. Such inaccuracies are mitigated with access to reliable audio sources, but that is not always the case.

A typical transcription session starts with picking up the song. The author hears it a couple of times and then proceeds to notate the song in his music notebook. Sometimes, he is in front of a digital piano or holding an acoustic guitar to help him locate the notes in the staff more easily (Fig. 1). After a few tweaks, he plays and sings the complete melody for proofreading. The first part of the transcription session is complete.

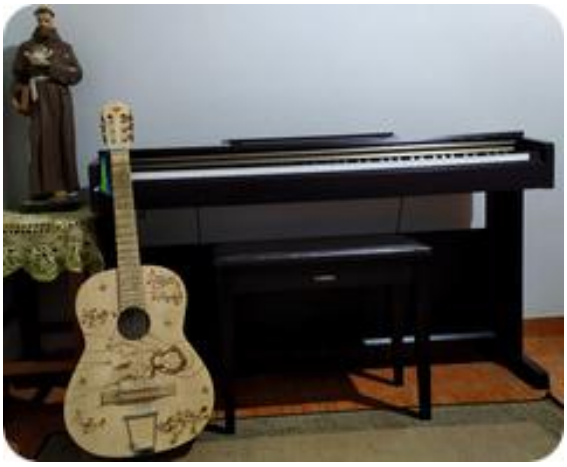


Figure 1: A typical transcription session environment.

Now the author needs to convert the songsheet sketched in his music notebook into a proper digital

version through computer music engraving, the art of drawing music notation at high quality for the purpose of mechanical reproduction. Although there are several TEX packages for music engraving, his approach goes towards LilyPond.

LilyPond's primary goal is to produce output comparable to professionally engraved scores instead of output that looks mechanical. Some of its features include:

- *Optical font scaling:* depending on the staff size, the design of the music font is slightly altered; this is a feature that the Computer Modern typeface is known for. As a result, note heads become more rounded, and staff lines become thicker.
- *Optical spacing:* stem directions are taken into account when spacing subsequent notes.
- *Special ledger line handling:* ledger lines are shortened when accidentals are nearby, thus enhancing readability.
- *Proportional spacing:* notes can be positioned in such a way that exactly reflects their duration. For example, with this setting, the space between consecutive quarter notes is four times greater than between consecutive sixteenth notes.

However, LilyPond solves only half of the task. The author also wants his liturgical songsheets to have text blocks with proper hyphenation, beautiful fonts, scripture verses and background images as well. This is a typical scenario where LilyPond's syntax and commands become a nuisance.

Surely, the best solution to this task is neither LilyPond or TEX, it is LilyPond *and* TEX. One can benefit from the best of two worlds: LilyPond with professional music engraving and TEX with its unparalleled text capabilities. So that is how the second part of the author's transcription session goes: he converts the songsheet sketch from his notebook into the corresponding LilyPond format and sets up the score in a TEX file.

When converting songsheets into the LilyPond format, the author typically uses a free software editor named Frescobaldi. This editor offers many templates to ease the writing of complex scores, as well as dictionary-based lyrics hyphenation and other engraving tweaks. The editor also features a score preview with point and click, which lets you find notes in the input by clicking on them. Observe that there is no need to handle paper size and margins in the LilyPond file, as such adjustments will be done later on, inside the TEX file.

For the TEX document, the author usually uses the standard `article` class with A4 paper and 12-point size font options. For margins, the `geometry` package is used. If the text is too long, the author also includes the `multicol` package; otherwise, a simple `minipage` suffices. Finally, the author also loads the `background` and `graphicx` packages to provide watermark features and image support, respectively.

The LilyPond magic inside these TEX documents happens thanks to a fantastic package named `lyluatex`, available out of the box from your typical TEX Live and MiKTEX installations. As the name implies, this package can only be used with the LuaLATEX engine. Some of its features include:

- LilyPond is used to compile music scores directly from within the LuaLATEX engine run. Music scores are created in real time, so shell escape is required during the compilation.
- Intelligent caching of engraved scores, avoiding recompilation when possible.
- Matching of layout and appearance to perfectly fit the scores into the text document.
- Comprehensive configuration through global and per-score options.

Another important feature is the automatic font handling, which is disabled by default. As this approach uses LuaTEX engine, system fonts are available and `lyluatex` can handle them as well by passing such metrics to LilyPond.

It is worth mentioning that this workflow can be further automated by using a template tool like TEXplate to generate the code boilerplate that handles both text and score, merging them into a coherent document structure.

So this is how a typical transcription session goes. There are always manual adjustments, usually minor, during proofreading, both in textual and music typography. It is an art, after all. The author will spare the reader details of such adjustments, but they do happen.

3 Booklets

A booklet is typically the name given to a very thin book with a small number of pages and sometimes a paper cover, giving information about something. In this case, a song booklet contains liturgical songs. Since almost no one in the author's community can read sheet music, he decided to just put the song lyrics in the booklets. Granted, it is rather unusual to spot sheet music in today's liturgical booklets, unless the reader is lucky enough to live near Rome and can attend the Holy Mass in Saint Peter's Basilica. The Vatican website contains lots of mass booklets for download.

Finding a good layout for a song booklet is challenging. The author did several experiments

throughout the years with page size and margins, font shapes and colours, number of columns and other typographical aspects, in the hopes of finding the perfect balance between aesthetics and ergonomics. A good booklet project for a parish has to be useful without taking the person's focus from the liturgical celebrations.

The author found A5 to be the best paper size for his booklets, as it is easier for people to hold during masses and saves on printing resources. An A5 is equivalent to half of an A4 paper, so this approach gets the four pages instead of the usual two from a typical printing.

For these TEX documents, the author again usually uses the standard `article` class with A5 paper and either an 11 or 12-point size font for options. He also enjoys fonts with round shapes, as a means to improve reading, so `bookman` is a good choice. For margins, the `geometry` package is used. The author found two columns to be a good balance between aesthetics and ergonomics, so the `multicol` package is also used. Since titles are added to songs, the fantastic `tcolorbox` package is loaded as well. And at last, but not least, the `graphicx` package is used to provide image support. A sample booklet is presented in Fig. 2.

Figure 2: A sample booklet.

The author rarely applies a cover, so all pages are available for content. The number of pages can be one, two or any multiple of four. For the final document, he uses the `pdfpages` package to help him distribute the A5 pages into a set of A4 pages. A simple Python script is used to organize the page order.

There is a reason for this specific page order: the author can produce song booklets without page cutting. He just needs to print the A4 papers, fold and group them. A very special stapler is used to prepare these booklets in a couple of minutes (Fig. 3). This workflow is surprisingly efficient.

Figure 3: A very special stapler.

These song booklets do require work and final adjustments, but the layout and content disposition are usually straightforward. However, there are songbooks that require a significant amount of rearrangement of elements as a means to achieve better aesthetics and ergonomics.

4 Final remarks

This article presented the author's typographical adventures on producing songsheets and song booklets with TEX and friends. It is worth mentioning that Gregorian chants can be typeset as well, using the `gregoriotex` package and a file containing the corresponding chant in the GABC format.

The author is interested in solutions for automating either all or part of the layout process, determining sizes and positions of visual elements. This field of research seems to lie at the crossroads between artificial intelligence and human-computer interaction. So far, the author is trying constraint-based methods. Preliminary results are promising, but there is still a long road to walk.

◇ Paulo Roberto Massa Cereda
São Paulo, Brazil
paulocereda (at) gmail dot com

Empowerment and teaching LaTeX

Astrid Schmölzer, Sarah Lang

Abstract

This talk addresses the question of how empowerment (and lack thereof) influences teaching and learning LaTeX, on the example of creating collaborative solutions between Humanities scholars and LaTeX consultants. It recounts the backstory behind the personas of Noob and Ninja. It teaches four lessons on empowerment in teaching LaTeX:

1. Empowerment is much more than providing motivation
2. Being welcoming
3. Focusing on teaching the basics extremely well
4. Empowering new users to find their own solutions

The backstory

After an unsuccessful attempt at typesetting a master's thesis in LaTeX in 2013, it took four more years until Astrid's LaTeX story finally started. As a Humanities scholar and archaeologist, there wasn't a lot of online support geared towards her needs. Nor were people willing to help her because, most likely, they were prejudiced against her for coming from the Humanities and not a technical field. Her encounters didn't go well. They tended to be aggressive or degrading in some way, both on the internet and in real life. This is why, in 2018 when the LaTeX Ninja blog launched, we soon partnered up as Noob and Ninja to post this story of rejection.[1] The blog post was welcomed very warmly by the (La)TeX community and a first article about it appeared in *TUGboat* in 2019.[2]

In our talk at TUG 2020, we explained typical typesetting needs of Humanities scholars using the example of an archaeological catalogue, which essentially requires setting up an environment which features non-floating images, tabular data and a description which can be a longer text. These catalogues are not exactly standardized, they can be very different from one individual to another and thus, of course, there are many possible LaTeX implementations for them. However, the typesetting solution chosen is not the point of this paper. Here, we want to draw more general conclusions about what can be learned from this scenario about empowerment in teaching LaTeX.

One part of this empowerment is to transform the term "newbie" or "noob" which is very often used in a negative way into a positive idea of a beginner of LaTeX (or any other technical area one might start as a Humanities student or scholar). The Ninja and the Noob stand for this positive way of using the term "noob" and we go back to the very meaning of the word itself: a beginner. And all (La)TeX users were beginners once.

Lesson No. 1: Empowerment is much more than providing motivation

In fact, it's mostly about eliminating or reducing sources of potential demotivation. After all, motivation doesn't last but demotivation does, so we want to avoid it like the plague. It also doesn't mean that we need to "entertain" new users. They can take on a challenge. They don't require a circus of fun tutorials. They just require tutorials which don't confuse them more than they help. They might need to be taught how to create a minimal working example to ask questions effectively on StackExchange. It might not be immediately obvious for them how the lipsum text seemingly appeared from nothing. They might need the pre-tutorial tutorial. You need completely different didactic approaches for teaching newbies than for other users.[3]

Lesson No. 2: Being welcoming[4]

Don't be imposing. Don't expect newbies to follow you like a hero. Empower them to create their own solutions but also don't stop talking to them because they decided to go for an implementation which you recommended they shouldn't use. The goal is to keep them using LaTeX. As long as they continue using it, they will get better. Only a user who stops using LaTeX is a "bad user". Failures are part of the learning process. Indeed, there can't be any learning without mistakes made. (This is also a reason why we need to teach coping with error messages, including psychologically!)[5]

But part of being welcoming also means to write that follow-up email if you said you would. About summing up recommendations given in oral form again in writing — because it's so easy to get lost with just one single typo or misunderstanding. Part of being welcoming could also be giving a newbie good keywords for a web search — after all, their search engine is probably not trained to spit out great LaTeX resources like yours probably is. Give suggestions on how to best find good search results, which terms or packages to look for rather than patronizing new users for "not looking up the problem on their own". Maybe they did. Maybe the problem is ridiculously easy in your eyes. If they didn't find a solution even though you think a quick web search yields tons of good results, don't assume they didn't look. Maybe they did but didn't find anything because

they didn't know how to formulate the problem (just as they might not know how to "correctly" formulate a problem for posting the question on a forum).

All this requires previous knowledge, and abstraction isn't easy, just because it comes naturally to you as an experienced user. Maybe they found a solution but didn't understand it or they didn't realize a search hit was relevant to them (or how it could be relevant to them). Also, don't bother with useless reprimands instead of answering the question. You are not in the newbie's shoes so it is not your place to judge them. Make sure to be welcoming instead.

Lesson No. 3: Focusing on teaching the basics extremely well

When teaching people from the Humanities, the first obstacle is the very different idea they have for why to use LaTeX and what to use it for. The Humanities are a complex gathering of fields with their own methods and research questions. Therefore, their interests in using LaTeX include a lot of possible requests.

Because of the problems and barriers the noob experienced, we decided to choose the production of an archaeological catalogue as a project to start with.[6] Knowing the basic outline for the needs of the Noob, the Ninja could start working and thinking about it. Every Humanities problem demands another field of basics — some are fairly similar to other research questions and requests for the use of LaTeX, while others are unique. So the basics you have to teach depend on the specific needs of your Humanities noob. Focusing on teaching is thus focusing on individual needs and on talking about the individual needs and concerns.

But it also means you don't need complicated solutions. Just make sure people get the basics right. Often this consists of a process of extreme simplification after the first brainstorming phase. Then you need to patiently and persistently reinforce the concepts.

Lesson No. 4: Empowering new users to find their own solutions

This can mean teaching which packages might be relevant for them because there is a whole flood of possibilities out there on CTAN.[7] As mentioned above, suggesting to new users which search terms typed into a search engine are likely to yield the desired results: Often, newbies cannot search for their own problems effectively because firstly, they're using much less targeted or relevant keywords, and secondly, the engine doesn't remember all sorts of useful online LaTeX resources for them, like yours probably does.

Finding your own solutions for a newbie means finding imperfect solutions and that's ok. Learning how to write one's own first command or environment is an important milestone on the path to LaTeX empowerment. This also relates to hammering the basics home. Small successes are essential for empowerment. And understanding the basics well is the best way of scoring small wins to be able to move forward.

Notes

[1] Astrid Schmölzer, Guest Post, Confessions of a LaTeX Noob, in The LaTeX Ninja Blog, 26 February 2019, latex-ninja.com/2019/02/26/guest-post-confessions-of-a-latex-noob

[2] Sarah Lang and Astrid Schmölzer, "Noob to Ninja: The challenge of taking beginners' needs into account when teaching LaTeX", *TUGboat* 40:1, 2019, pp. 5–9, https://tug.org/TUGboat/tb40-1/tb124lang-newbie.pdf.

[3] For more detail on the didactics, see Sarah Lang's article in this issue, "Didactical reduction versus references", pp. 173–175.

[4] Discussed in detail in the "Noob to Ninja" *TUGboat* article cited above.

[5] See for example the posts "Learning to program: What to do if the program doesn't compile", latex-ninja.com/2020/01/12/learning-to-program-what-to-do-if-the-program-doesnt-compile, and "Learning to program: Failing fast and error messages", latex-ninja.com/2020/05/31/learning-to-program-failing-fast-and-error-messages.

[6] We won't go into detail here, but in this post are some of the insights we learned from the experience: latex-ninja.com/2020/05/04/latex-for-archaeologists-an-archaeological-catalogue-using-latex.

[7] During the conference, a discussion started on whether it would be possible to filter packages by utility.

◇ Astrid Schmölzer, Sarah Lang
the.latex.ninja (at) gmail dot com
https://latex-ninja.com

Didactical reduction versus references: How to better teach LaTeX

Sarah Lang

Abstract

This paper discusses didactical principles (such as didactical reduction) and their relationship to common modes of technical (and (La)TeX) knowledge transmission, mainly the genre of technical references and documentation. It leverages ideas from personal development for suggesting new, more didactically suitable modes of knowledge transmission for (La)TeX education. It discusses how teaching concepts (like didactical reduction) and common modes of knowledge transmission (such as references and documentation) could be reconciled in LaTeX teaching contexts.

The problem: Informal knowledge about how to act in a learning setting and how to procure the knowledge one needs is often linked to different forms of privilege. Using references to acquire knowledge requires a relatively big amount of tech literacy or familiarity with the medium. People with non-tech backgrounds often lack this tacit knowledge. This paper proposes ways to provide different sorts of LaTeX education to different types of learners, using the efforts made on the LaTeX Ninja blog as an example.

This paper is not an exact replica of the contents of the talk. The paper contains an example discussion of Kopka and Daly's *Guide to LaTeX* which came from the discussion after the original talk; and the original talk contains information about the LaTeX Ninja blog and other examples which will not be discussed further in the paper.

1 What is didactical reduction?

Didactical reduction is the art of reducing unnecessary detail.[1] By this, I don't mean at all that the difficulty level of materials necessarily needs to be reduced. Dumbing things down so much that there is no content left is not good practice either, and reducing information to the purely superficial is not at all what I mean. One can present difficult material but still leave out detail to make the contents more digestible didactically. This entails that producers of didactical materials need to become aware of their tacit knowledge and "tech privilege" before writing up learning materials. For example, passive knowledge of packages becomes a dominant issue when numerous packages are included in tutorials — packages which aren't part of the topic and could have been left out. This overcomplicates the subject and provides examples which are far more than minimal.

2 Different starting points: "Tech privilege" and tacit knowledge in the (La)TeX learning setting

I argue there exists "tech privilege" which is a form of privilege analogous to other forms of privilege, such as male or white privilege, but concerns the subject matter of technology. It is known nowadays that we are taught lots of tacit knowledge implicitly according to certain stereotypes. For example, studies involving cross-dressing in small children, e.g., female toddlers dressed in stereotypically male clothes and vice versa, have shown that adults are conditioned societally to pass the "female passing" children dolls and such, whereas they give "male passing" children stereotypically "male connotated" toys, which tend to teach more logic, technical and mathematical skills. Such early conditioning causes children's "talents" to develop much more in the area favoured by society, generating an apparent divide that we all know: It indicates that men are supposed to be better at logic and maths, whereas women are better at languages — a wholly incorrect stereotype according to science!

Children (and adults too) thus get more praise for engaging in stereotypically gender- and class-suited activities. They will also be trusted more to be successful and apt at those. It is known that praise and expectations, as in the so-called Pygmalion or Rosenthal effect have an influence on learning.[2] This means that those who already have a certain "tech privilege" or are in a socially accepted position as a tech nerd have a much easier path learning tech knowledge than those who lack this "tech privilege". Also, those with "tech privilege" will implicitly learn certain skills pertaining to tacit knowledge — say, knowing how to post a successful question on StackOverflow or the simple fact of what a Lorem Ipsum is.

A person without a background in technology might never have come across such concepts which makes their learning process all the more difficult if learning resources don't provide this information. To those who aren't already part of a larger tech community, typical technical modes of knowledge transmission, such as references and documentation, are largely inaccessible.

3 The ability to abstract and the "superuser mindset"

Apart from that, references often require a large amount of abstraction which is also a skill outsiders might not yet have. They don't know how to sum up their problem in an abstracted way. For example, in the talk, I mentioned that a newbie might not realize

their problem is spacing. They might just think, "I want this page to look like I know it from MS Word." They also might not be familiar with non-GUI applications, having seen only applications which are extremely targeted to fulfilling a user's (presumed) needs, giving them a user experience which requires no background knowledge at all. Moving from a passive user experience to — what we could call — a "superuser perspective" is a big step which is not at all an obvious or easy path for a newbie without a tech support network or in-person teaching.

4 The difference between reference and didactical reduction, with the example of Kopka and Daly's *Guide*

I was asked about my opinion on the generally well-liked reference of Kopka and Daly.[3] Therefore, I have chosen it as an example on which I can illustrate my point about references versus didactical reduction. I was asked whether I would recommend it and my opinion is: as far as references go, this a go-to resource — but it is still a reference. My argument is that for effective and efficient teaching, we need to detach teaching resources and technical reference documents. Kopka and Daly explicitly say their guide is a mixture of both, which is exactly what I take issue with.[4] Thus, I would not fully recommend it. This does not mean I don't like the *Guide* — I just think it was written under a different paradigm from the one I invite the (LA)TEX community to use in the future.

Let's examine this in the context of the *Guide*. The first thing we notice: it's over 600 pages long. So, either you're not expected to read it all (i.e., it is a reference), or it's going to be a long read. I think it's a good beginner's reference but I also find there's an inherent problem with references. I would expect a beginner's text to only cover things that they will ubiquitously need, so that's either "theory" or a focus on the absolute basics. For example, if you know how a document class works, why would a reader be shown the example of letters which they might not even need? If they wanted to learn about them, they could just look it up with a quick web search. Thus, I think there would be lots of room for cutting out detail.

A suggestion for improvement would be to perhaps write a beginner's part with things actually everybody will need and then diverging into "specializations".

For example, as has been addressed elsewhere during the TUG 2020 conference, TEX people tend to assume everybody needs math. But I never need math. These little "divergences" might be starting points to find elements which a general reference should exclude or put into an appendix. Kopka and Daly for example put "Programming with LATEX" into the appendix which I think should go into the main section, just like the text on "Error messages". On the other hand, in my opinion, Kopka and Daly should have put lots of the material presented in part II "Beyond the Basics" into the appendix. A resource following the paradigm of didactical reduction would put all the relevant background information ("theory") into the main part and move the information about packages or special purpose document classes into the appendix.

References are not usually didactically suitable. Apart from the format being less than optimal for teaching purposes, references also lack certain elements which are relevant to teaching, i.e., explaining theoretical backgrounds.

5 Leveraging concepts from personal development: Different goals

In theory, learners from every different skill level and background would each need their own learning resource, which is why I suggest that complete beginners should get extremely short resources with lots of practical applications, such as my attempt at a "3 minutes to LATEX" tutorial.[5]

On the other hand, it can often be difficult to find information on the skills which lead to true advanced LATEX skills or LATEX mastery — because this also includes "soft skills" such as knowledge about the publishing industry, an understanding of typography or how to monetize your LATEX skills. Leaning on different concepts of skill acquisition of expertise from personal development, I have argued that mastery might not be for everybody, which is why we shouldn't make it the goal of learning.[6] Josh Kaufman's *The First 20 Hours: How to Learn Anything ... Fast*, for example, points out that with popular skill acquisition discussions like the so-called "10,000 hour rule", the implicit assumption was made that everybody wants to achieve real mastery.[7] Kaufman argues that is quite the misconception; in reality, many people just want the skill level equivalent of being able to strum along with a four-chord song on the guitar. Certainly some will want to achieve mastery, but by no means all.

The (LA)TEX community should acknowledge this diversity not only of background in different learners, addressed in the discussion on tech privilege and tacit knowledge, but also the diversity of possible goals a (LA)TEX learner might have in mind.

Notes

[1] I have addressed the topic in the following blog posts:
latex-ninja.com/2018/11/05/on-didactical-reduction-especially-in-the-dh
latex-ninja.com/2019/01/12/didactical-reduction-part-ii
latex-ninja.com/2019/04/14/improve-your-teaching-10-simple-tricks

[2] The effect describes a self-fulfilling prophecy that learning effects get better with higher expectations and worse with lower expectations.

[3] Helmut Kopka and Patrick Daly, *A Guide to LaTeX*, 4th edition, Addison-Wesley Professional, 2003.

[4] "This *Guide* is meant to be a mixture of textbook and reference manual."; page 10, Kopka and Daly.

[5] latex-ninja.com/2018/12/11/jumpstarting-learn-latex-in-3-minutes

[6] In the talk, I also brought up the concept of the Minimum Effective Dose (MED) from Tim Ferriss' *The 4-Hour Chef*. This was discussed in our earlier *TUGboat* article: Sarah Lang and Astrid Schmölzer, "Noob to Ninja: The challenge of taking beginners' needs into account when teaching LaTeX", *TUGboat* 40:1, pp. 5–9, https://tug.org/TUGboat/tb40-1/tb124lang-newbie.pdf.

[7] Malcolm Gladwell's book *Outliers* made the so-called 10,000 hour rule popular. However, please note that this was an incorrect interpretation of research done by Anders Ericsson. The best resource about the science of expertise and skill acquisition using deliberate practice is Ericsson's *Peak: Secrets from the New Science of Expertise*. A commenter also suggested the book *Range: Why Generalists Triumph in a Specialized World* by David Epstein.

◇ Sarah Lang
the.latex.ninja (at) gmail dot com
https://latex-ninja.com

LaTeX-on-HTTP: LaTeX as a commodity web service for application developers

Yoan Tournade

Abstract

Although LaTeX is widely used in academia and education, only a few developers use it to create PDFs in web applications or IT systems. This seems strange considering that many developers are well exposed to these academic and scientific milieux — and that many recognize LaTeX for its superior typesetting qualities.

In this paper, we try to answer two questions:

- Why use of LaTeX in IT systems is not widespread — and not even common?
- How can we change this state of affairs?

In this article we give an overview of LaTeX-on-HTTP (github.com/YtoTech/latex-on-http), an attempt to commodify and simplify LaTeX use in web applications or IT systems, and to ease its adoption by modern developers.

1 Introduction

LaTeX has its indisputable niches, like academic and scientific publications. Thanks to web projects like Overleaf (overleaf.com), it has also become more readily accessible for the lay computer user to complete common editing chores.

I found it strange then that one of the populations most exposed to LaTeX, and also the most likely to recognize its merits, *the computer engineers*, do *not* use LaTeX in the applications they develop — and that they may not even think to do so.

Currently a modern application developer that needs PDF output for a project will find an abundance of solutions by searching the web:

1. HTML/CSS to PDF converters, e.g., wkhtmltopdf (wkhtmltopdf.org) and WeasyPrint (weasyprint.readthedocs.io);
2. instruction-based PDF generators, e.g., PDFKit (pdfkit.org) in Node, and FPDF (fpdf.org) in PHP;
3. headless calls to browsers or office software, e.g., Puppeteer (github.com/puppeteer) is an example of a headless API to run Chrome/Chromium browsers with no GUI.

Browsing a couple of *Stack Overflow* entries will give our hypothetical developer copy-paste ready initial working code, in his language of interest. LaTeX will not appear in the results; and may well not pop up in our developer's mind as an eligible solution.

All these tools provide decent results, but we argue they are far from the typesetting quality attainable by LaTeX. While we can understand the prevalence of the first class of solutions above by the opportunity to leverage existing HTML and CSS code, this does not explain why LaTeX would be virtually absent from the common set of solutions.

2 Why LaTeX is uncommon in modern applications for PDF creation

We explain this state of affairs from three viewpoints: techniques, customs and knowledge.

2.1 Technical barriers: the not-so-accessible LaTeX runtime

It is easy enough to install a LaTeX distribution on a personal computer. Modern browser-based services like CoCalc (`cocalc.com`) [4] and Overleaf have even removed this need and drastically reduced the time required to first interact with LaTeX [2, 3]. Tools such as MathJax (`mathjax.org`) and LaTeX Base (`latexbase.com`) [1] have even demonstrated that LaTeX (or a subset) can be run in a browser.

However the requirements of an application developer are different: he does indeed want convenience, but even more importantly he needs reproducibility and scaling — and he certainly does *not* want to battle with his infrastructure team to explain why it would be pertinent to add many hundred of megabytes of complex runtime requirements to generate PDFs in their applications. Many developers do not even fully control their runtime environment (restricted cloud, etc.).

When the developer has only a few hours, or at best days, to automate PDF creation, he will shy away from the complexities of adding the LaTeX runtime in his application — he has to go for the safer and more recognized solutions.

2.2 Cultural glass walls: mental buckets of use cases

Developers, often coming or having been exposed to academic and scientific milieux, know of LaTeX and recognize its typesetting superiority. They are the people who might tease one of their mates for *not* using LaTeX in their latest technical publication.

However, for them LaTeX is in a mental bucket that excludes their web application development or IT system integration activities. More often than not, they will not think about LaTeX for making a PDF document in their application: a glass wall of custom separates their need from the LaTeX solution.

2.3 Imperfect knowledge bases: learning by copying

Even if our developer considers LaTeX as a prospective solution, he must then learn about its implementation.

As developers, we try not to reinvent the wheel with each task. We search and we find shared knowledge about recommended tool usage and common patterns. Generally, we use readily available and copy-pasteable code samples for our needs; and tweak our custom solution from there.

Our developer could easily find a proper LaTeX template to start from for his PDF application — there are great resources on the web for that. But then? Our developer will need the code to compile their LaTeX template to a PDF, which they will not easily find on the web. Compile a LaTeX document on my computer or on the web? Easy. Compile a LaTeX template from a web application — either from a server or from any browser[1] — in a couple of lines of code and without adding potentially problematic requirements? Not so easy.

Few pertinent examples are available to lead our developer to a way to implement LaTeX in his application. If he has the time to be curious and is up for a challenge, he can find a path [5, 6]. This is especially so in the niches where the need for very high-quality or specialized document creation will justify the cost of bringing LaTeX into the infrastructure. But this is not the typical case that interests us here: we consider the common modern web developer that at first just wants to create a rather simple PDF document in his application.

3 Introducing LaTeX-on-HTTP

LaTeX-on-HTTP first emerged when I needed a way to compile LaTeX documents to automate invoicing, without installing the whole LaTeX stack.[2]

The basic requirements for LaTeX-on-HTTP were rather straightforward: take the TeX Live runtime, put it on a server, and add an HTTP-based API between the user and the server so we can pass the files to be compiled. *Voilà* — now users just need Internet access to generate their PDF documents; they do not even need to know that they are using LaTeX.

[1] And considering the disparate array of web browsers, this is not the least of requirements.

[2] After converting my quotes-and-invoices consulting templates to LaTeX, I was glad of the result, but I wanted my other team members to be able to make these PDF documents without them having to install gigabytes of additional software, or me having to support them for managing missing CTAN packages.

3.1 HTTP: the web *lingua franca*

The HTTP API is an important part of the solution; it gives us several desirable properties:

- as HTTP is ubiquitous in modern applications, the service is accessible from most of the technical stacks: no more need for complex runtime requirements, a simple HTTP client suffices;
- the HTTP protocol is well-known to most developers;
- it clearly advertises the solution as intended for automation and integration.

LaTeX-on-HTTP is not the first solution to put the LaTeX stack behind an HTTP API,[3] but it may be the first designed with developers as first-class users.

3.2 Hello world: specifying a compilation job with JSON

Let's now present how to use the LaTeX-on-HTTP for compiling documents in applications.

The following JSON[4] code is sent in a POST HTTP request to the `/builds/sync` endpoint:[5]

```
{
"compiler": "lualatex",
"resources": [
    {
        "main": true,
        "content": "\\documentclass{article}
        \\usepackage{graphicx}
        \\begin{document}
        Hello World
        \\includegraphics[width=5cm]{logo.png}
        \\end{document}"
    },
    {
        "path": "logo.png",
        "url": "https://www.ytotech.com/static/
images/ytotech_logo.png"
    }
]
}
```

(The line break in the `url` value is editorial.)

This request will return a PDF file if the compilation succeeds. If there is an error, the API will return a JSON payload including the compiler logs.

As we can see, we pass two main things to the LaTeX-on-HTTP endpoint:

- a set of resources — or source files — to be compiled, with the path specified for each;
- the engine selected — and other potential options — to control the compilation environment.

We may also remark that we can use different means to transfer the resources to be compiled: here, by passing the string content directly, or by providing a url pointing to a file. We can imagine and provide several other ways, for convenience and for supporting various use cases.

It is essential to normalize as much as possible the LaTeX compilation job input for ensuring the reproducibility of the process and to provide further capabilities, such as caching of input resources or output documents.

3.3 Real-life example: editing a letter

A developer can use this API to construct a real-life application. Let's say our developer wants to edit a personal letter from a bank IT system, notifying customers of their account opening. Using Python as our language, the code could look like the following:

```
# Open and read the template LaTeX file.
with open("opening-letter-template.tex") as f:
    template_str = f.read()

# Replace the dynamic content.
template_str = template_str.replace(
    "<<letter-title>>, "Your account...")
template_str = template_str.replace(
    "<<letter-body>>, "Dear Mr...")

# Loads a binary image file.
with open("duck_logo.png", "rb") as f:
    logo_binary = f.read()
    # Convert to base64.
    logo_b64 = base64.b64encode(
        logo_binary).decode("utf-8")

# Generate the PDF file,
# using an HTTP client (requests).
r = requests.post(
    "https://latex.ytotech.com/builds/sync",
    json={
        "compiler": "pdflatex",
        "resources: [
            {
                "main": true,
                "content": template_str
            },
            {
                "path": "logo.png",
                "file": logo_b64,
            },
```

[3] ShareLaTeX/Overleaf's CLSI (`github.com/overleaf/clsi`) and Andrey Lushnikov's LaTeX-Online (`github.com/aslushnikov/latex-online`) are significant open source precedents.

[4] JSON (`json.org`) has become a dominant force in data serialization languages in web applications and APIs, rivaling or surpassing XML.

[5] You can easily try a similar example on the command line of your computer by using `curl`. A snippet is available on the project page: `github.com/YtoTech/latex-on-http`.

```
        ]
    }
)

# Save the PDF output.
with open("opening-letter.pdf", "wb") as f:
    f.write(r.content)
```

In this example, the string interpolation is managed directly with Python: the templating could have been done with LaTeX tools, but it is easier and faster for our developer to inject the dynamic content in his main programming language.[6]

We can also note that we used another resource transfer mechanism: the local image file is passed as a binary object (with a base64 encoding required by the string-based JSON API).

From this simple base example, our developer can easily inject his application data in the letter to be edited; he can then take the resulting generated PDF to return it in a web page, in his own application API and/or to save it in a persistent storage system for later use.[7]

4 The road ahead

LaTeX-on-HTTP is still a project in development; it must be enhanced to meet more developers' needs and use cases. We have already received excellent feedback, but we actively encourage and need prospective users to try it and let us know their observations and feelings.

Several subjects have been left aside in this introductory paper — or are still not properly dealt with by the current LaTeX-on-HTTP implementation:

- providing an asynchronous compilation endpoint;
- bringing alternative output modes to the PDF binary, like DVI for advanced uses or PDF.js (`mozilla.github.io/pdf.js`) for universal web embedding;
- securing and isolating the compilation jobs;
- advanced and reliable run configurations;
- caching output documents for dealing with duplicated jobs;
- caching input resources for minimizing bandwidth usage;
- discovering and managing the instance capabilities (fonts, packages, etc.).

In addition, to further aid adoption and reduce the time to first interaction, library wrappers and utilities need to be provided for the popular programming languages.

The next important work, however, is not in further development but rather writing high-level documentation and presentation material.

5 Conclusion

By bringing the runtime requirement to use LaTeX to a familiar HTTP API, we remove the main friction preventing developers from adopting LaTeX in their applications for PDF document creation.

To bolster the prospective success of LaTeX and LaTeX-on-HTTP as a reference solution for developers, we must address the cultural glass wall and knowledge barriers by:

- publicizing this usage of LaTeX in web applications and IT systems with ready-to-use demonstration code and libraries;
- publishing attractive documentation and presentations of the LaTeX-on-HTTP API.

Only then can we hope to see better typeset PDF documents in our daily applications as a result.

References

[1] G. Aye. Introducing LaTeX Base. *TUGboat* 37(3):275–276, 2016. `https://tug.org/TUGboat/tb37-3/tb117aye.pdf`

[2] S. Lang, A. Schmölzer. Noob to Ninja: The challenge of taking beginners' needs into account when teaching LaTeX. *TUGboat* 40(1):5–9, 2019. `https://tug.org/TUGboat/tb40-1/tb124lang-newbie.pdf`

[3] P. Łupkowski. Online LaTeX editors and other resources. *TUGboat* 36(1):25–27, 2015. `https://tug.org/TUGboat/tb36-1/tb112lupkowski.pdf`

[4] H. Snyder. SageMathCloud for collaborative document editing and scientific computing. *TUGboat* 38(1):44–47, 2017. `https://tug.org/TUGboat/tb38-1/tb118snyder.pdf`

[5] B. Veytsman, L. Akhmadeeva. TeX in the GLAMP world: On-demand creation of documents online. *TUGboat* 31(2):236–239, 2010. `https://tug.org/TUGboat/tb31-2/tb98veytsman-glamp.pdf`

[6] B. Veytsman, M. Shmilevich. Automatic report generation with Web, TeX and SQL. *TUGboat* 28(1):77–79, 2007. `https://tug.org/TUGboat/tb28-1/tb88veytsman-report.pdf`

[7] U. Ziegenhagen. Combining LaTeX with Python. *TUGboat* 40(2):126–128, 2019. `https://tug.org/TUGboat/tb40-2/tb125ziegenhagen-python.pdf`

[6] In a more complex case, we could have used a dedicated templating engine, such as Jinja2 (`github.com/pallets/jinja`). For more about combining LaTeX and Python, see [7].

[7] Complete demo code can be found for Python at `github.com/YtoTech/talk-TUG2020-LaTeX-on-HTTP` and in JavaScript at `github.com/YtoTech/latex-on-http-demo`.

◇ Yoan Tournade
Soubeyrac
Le Laussou, 47150 France
y (at) yoantournade dot com
https://yoantournade.com

Using Overleaf for collaborative projects: First impressions and lessons learned

Boris Veytsman

1 Introduction

The COVID19 pandemic has changed many things. Among them are the ways scientific papers are written. In the past, an author could assume to have physical meetings with the coauthors, to exchange written pages, ideas and thoughts. Unfortunately today these meetings also may mean exchanging virion particles. Thus many collaborations now rely exclusively on virtual meetings. Anybody who has participated in such meetings knows they cannot compare with the lively back and forth of in-person interaction. This puts a heavy burden on the technology involved: we need to compensate for the deficiency of the virtual collaboration with our tools.

In the first months of the pandemic I participated in two completely virtual collaborations, resulting in the papers [1, 2]. One paper has seven coauthors, the other has four. The proper organization of the writing process was therefore very important for us.

Collaboration tools should have several important features. At a minimum they should track the changes in the manuscript and the contributions by the different authors. They should be able to typeset the manuscript at each stage of the preparation. Ideally they ought to provide a way for the coauthors to exchange metacomments about the text.

In the past my tool of choice was the combination of a local TEX installation (TEX Live) and a version control system (*git*, *svn*, *cvs*). This is a very good solution for typesetting and change tracking. On the other hand, it is not a good solution for metacomments, unless one was willing to put the paper on GitHub and use their issue-tracking system. Frequent physical meetings between the coauthors somewhat alleviated the lack of metacomments: the coauthors could always give their comments in person.

The big disadvantage of this workflow is the rather demanding requirements on the coauthors. They need to be not only comfortable with TEX (this is something to be expected from math and science people); they also need to have and maintain a local TEX installation *and* be familiar with a version control system. Personally, I think the ability to use version control is an essential skill for anybody working professionally with texts. Unfortunately, the reality is that this knowledge is rare outside the programming community (and even in this community it is often rudimentary; see Figure 1).

Figure 1: *Git*, from `https://xkcd.com/1597/`

Overleaf (`overleaf.com`) was suggested as a collaboration tool. We used it for the papers mentioned above. In this paper I discuss our experience and the lessons learned.

2 Overleaf workflows

There are several ways of working with Overleaf. First, one can use its native interface; each coauthor logs into the site, uses its online editor to create TEX files, uploads the images for the figures, and uses the cloud TEX installation for compilation. Some of my collaborators chose this workflow. However, it was not convenient for me. As a person who has spent countless hours with my trusty Emacs, I cannot imagine writing texts with anything else (an obligatory aside: I have complete respect for my *vi*-using friends). Also, I did not trust the Overleaf version control system, so I wanted to use my own.

Fortunately, Overleaf provides another workflow (currently only for paid accounts — but see below). Namely, it can serve as a *git* remote server. You can do a `git clone` for an Overleaf repository, and then use the familiar `git push` & `git pull` commands to sync the local *git* repository with the one at Overleaf. This workflow has the additional advantage of being very flexible; some coauthors can use the native Overleaf interface, while some coauthors can use *git*, and the two are smoothly integrated.

It should be noted that Overleaf offers yet another workflow: integration with GitHub rather than a local user's repository. However, we considered this scheme too complex, and we did not use it.

3 Overleaf access control

Whether the resulting paper is freely distributed or not, most authors would not like to show the world their unfinished work, with all its embarrassing errors, vague ideas postponed for the future papers, heedless statements, etc. Thus it is crucial for a collaboration tool to support defining who has the right to read and edit the manuscript.

Overleaf has two modes of access control. The first mode, available for both free and paid accounts, is based on link sharing. A project has a link for read/write access and a link for read-only access. Anybody in possession of these links (each being a long combination of random letters and digits) has the corresponding rights.

In the second mode, one can give the rights (read/write or read-only) to the chosen Overleaf users (who must log in to Overleaf to participate). The maximum number of collaborators for each project depends on the account tier: from one collaborator for free accounts to unlimited collaborators for "Professional" ones. It is important to note that the limit is determined by the tier of the project owner: if the owner has a "Professional" account, any number of other collaborators can have free accounts. The same is true for the *git* integration discussed above: if the project owner has a paid account, all other collaborators can use *git*.

In general, the Overleaf access control model seems to be mature and corresponds to the current industry best practices. The possibility to use third parties' credentials to login to Overleaf (IEEE, Google, Twitter, and ORCID) is also a convenient feature.

4 Overleaf TEX

To tell the truth, I had my doubts about the Overleaf TEX installation. My TEX is sometimes rather complex, and I wondered whether Overleaf could tackle it. These doubts proved to be wrong. I was quite impressed by the fast and correct typesetting on Overleaf servers.

The installation is rather complete. Currently the user can choose between *pdflatex*, *xelatex*, and *lualatex*, and the site has TEX Live installations from 2014 to 2019. Both *bibtex* and *biber* are supported.

In general, Overleaf has solved the problem that has always bugged novice TEX users: administration-free maintenance of the installation.

5 Comments system

In the old times manuscripts and books had large margins because coauthors, editors, and sometimes even readers used them for metacomments (Figure 2).

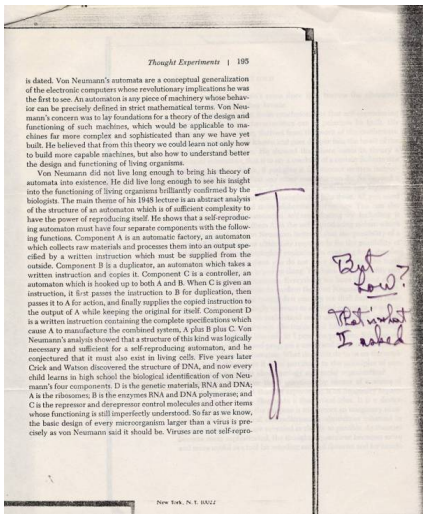


Figure 2: Annotations by Linus Pauling made to a passage in the book *Disturbing the Universe*, by Freeman Dyson, from `http://scarc.library.oregonstate.edu/coll/pauling/dna/notes/unpb17.1-automata.html`

When word processors appeared, they featured "comments": a user could click on a document and insert a special text, which was not part of the main flow, but a note for other authors. Web collaboration tools kept this feature and added integration with e-mails, messaging and task recording software; the comment could be "assigned" to a coauthor as a new task, and the assignee would be notified electronically with the corresponding record made in the task list.

TEX (or LATEX) by itself lacks this sophistication. Of course, one can add comments in the code using, for example, the percent sign convention, as in

```
\begin{equation}
 2\times 2 = 5    %%% Are you sure?
\end{equation}
```

Unfortunately, sometimes these comments are overlooked by coauthors. For better visibility, LATEX packages like *todo* may typeset them in the PDF output, for example, as marginal paragraphs.

The use of version control servers like GitHub allows one to employ a modern issue-tracking system which has all the functionality of the integrated comments, including notifications, assignments, etc. Moreover, the issues become part of the version control archive, so one can look at the history and establish that the given change in the manuscript was made by author A as a response to the comment made by author B — complete with the dated history of all responses.

For a user accustomed to these goodies, the Overleaf system of comments seems to be rather old-fashioned. Overleaf allows the users to create

a "comment" in the style of early word processors. These comments can be answered and "resolved". However, they are not shown in the PDF, cannot be addressed to a specific coauthor, and are not integrated with emails or the version history. While relatively easy to use, these comments are probably not adequate for a sophisticated user.

6 Overleaf version control system

An important feature of a collaboration tool is the ability to establish who wrote what, when and why. This is one of the tasks for a version control system. A version control system keeps track of the changes in the files, noting their authorship, dates, and, through commit messages, the reason for change. Ideally, it allows easy mixing of versions; a good program should understand a command like "Please delete all changes made in June, but keep the ones made in July".

The native version control system in Overleaf can do some of these tasks. It allows the user to compare the states of the document at the different stages of editing, find the changes, establish their authors and download the files at a previous version. This makes the work with Overleaf easier than with typical office software.

Unfortunately, the Overleaf version control system lacks branching capabilities. This means that major changes in a project require copying and renaming files — something that a version control system is intended to prevent. I thought initially that only the Overleaf interface to version control was deficient, and I could use branching in my *git* repository, and then synchronize with the Overleaf one. I was wrong; sadly, Overleaf allows only one branch to be uploaded to its repository.

Overleaf also lacks tags, and does not allow one to push them from the local *git* repository.

Reverting to a previous version is not a self-evident task in Overleaf. One of the coauthors inadvertently deleted a large portion of our TEX file in Overleaf. We found that the simplest way to recover was to use the tools provided by *git* on a local repository.

I was able to perform non-trivial merging of two different versions only by downloading them to the local computer, and using the tools there.

Overleaf version control system has "labels". They are mapped into *commit messages* rather than *tags*. They are also not enforced, so most changes in the files are not commented (in contrast, most version control systems refuse to accept commits without comments).

Of course, many *git* users complain it is too complex. The simplified version control of Overleaf might be a clever decision. While a sophisticated version control system allows an advanced user to perform many non-trivial tasks, it is indeed complex exactly because it has a rich feature set. An ideal system would be the one that works simply for a novice, but can do complex things. Clearly, designing such a system is not a trivial task. On the other hand, Overleaf solved the highly non-trivial problem of creating an intuitive interface for TEX — maybe it can create an intuitive interface for *git* as well?

7 Conclusions

Overleaf as a collaboration platform is a viable solution for coauthoring scientific manuscripts. Its strong features are a good access control model and underlying TEX installation. On the other hand, its system of metacomments can be improved, and its version control system is probably too simplistic.

Acknowledgments and disclaimers

I am deeply indebted to my colleagues who allowed me to observe their work with Overleaf.

Being a TUG officer, I acknowledge the generosity of Overleaf in their support of the organization through their institutional membership and several donations, including their help with TUG 2020, especially the LATEX workshop held there. The work described in this paper was done using a Professional level account, paid by Chan Zuckerberg Initiative.

The opinions in this article belong to me, and do not necessarily reflect the opinions of TUG, Chan Zuckerberg Initiative or George Mason University.

References

[1] G. Huber, M. Kamb, K. Kawagoe, L. Li, B. Veytsman, D. Yllanes, and D. Zigmond. A minimal model for household effects in epidemics. *medRxiv*, 2020. doi:10.1101/2020.07.09.20150227

[2] S. Satish, Z. Yao, A. Drozdov, and B. Veytsman. [A paper under blind review]. In *[Not disclosed]*, 2020. We were asked to suppress all information about the paper until it is reviewed.

◇ Boris Veytsman
Chan Zuckerberg Initiative
& George Mason University
& TEX Users Group
borisv (at) lk dot net

The Island of TEX: Developing abroad, your next destination

Island of TEX (developers)

Abstract

The Island of TEX is a collaborative effort to provide a home to community-based TEX projects. This article discusses the Island's long-term goals and how the worldwide community can come aboard and help the organization enhance the TEX experience for everybody, from newbies to power users.

1 The beginning

Once upon a time, a group of friends decided to create an organization as a means to systematize and concentrate our efforts in developing the TEX ecosystem. Thus, the Island of TEX, a collaborative endeavour to provide a home to community-based open source TEX-related projects, was finally charted in the TEX world map. Island residents include Paulo Cereda, Ben Frank, Brent Longborough, Marco Daniel, Nicola Talbot, and Enrico Gregorio.

So far, the Island holds four main groups of projects: Docker images, programming libraries, assorted tools, and TEX editors. We plan more groups for the near future.

2 Docker images

The first group to be discussed refers to Docker images. Docker is an open platform for developing, shipping, and running applications, enabling a clear decoupling of applications from infrastructure.

A Docker image itself is similar to a snapshot of a lightweight virtual machine. When running an image, it shares the same kernel as the host system but provides a complete and independent infrastructure of operating system and software packages, as well as configuration files and environment variables.

The island provides Docker images for TEX Live repositories. It is important to note that we also provide the necessary tooling to execute common helper tools. We have two groups: *historic*, which, as the name implies, contains releases from 2014 on, and *latest*, which refers to the current stable release plus all updates available as weekly snapshots.

For every TEX Live release, from historic to latest, we provide four flavours: binaries only, binaries and documentation, binaries and sources, and the full pack, binaries, sources and documentation. When in doubt, choose binaries only and only pull the larger images if you have to. Keep in mind that sources and especially documentation files do add a significant payload. If you need an image for an older TEX Live feel free to file a feature request. We might consider adding older distributions if there is a user base.

The Island also provides Docker images for the ConTEXt distribution (full installation with all modules). It also provides the necessary tooling to execute common helper tools. We have three groups: MkIV current (updated monthly), MkIV beta and LMTX (updated weekly). If you need a different image for ConTEXt as well, feel free to file a feature request.

3 Programming libraries

The second group to be contemplated refers to programming libraries. The Island currently provides three libraries: a SyncTEX parser, a TEXdoc API and a TEX log analyzer (which will be omitted here due to its pre-alpha stage).

3.1 SyncTEX parser

SyncTEX is a synchronization technology supported by all the major engines. It maps the input to boxes and point positions in the output. Therefore, it can be used for forward and backward synchronization, that is, finding the appropriate point in the output PDF for the current input line, or finding the respective input line for the current point in the output.

While collecting ideas for other tools, we came to the conclusion that there is no comprehensible and up-to-date structure definition of a SyncTEX file, so we started to implement a parser, partially as an exercise to know the file format and partially to provide some structured representation to the community.

It is important to note that we have not looked at the official SyncTEX parser while implementing this component and we do not intend to. We read only the man page and started to look at real world examples. Hence this is neither official nor guaranteed to parse all SyncTEX files. That said, you are most welcome to open issues and merge requests, contributing test cases where it fails.

3.2 TEXdoc API

The next library in the Island is an API for the `texdoc` command line interface. Hence, it is only one layer of abstraction; we currently require the presence of `texdoc` on the target machine.

The API provides two classes: the `Entry` class which represents an entry in `texdoc`'s list of results and the `TeXdoc` class, presented here, which is a singleton, providing an interface to the actions of the command line utility. The most important methods and attributes of `TeXdoc` are:

`isAvailable` to check whether `texdoc` is in the system path,

`version` to get the version string of the `texdoc` installation,

`getEntries` to get a list containing all entries from the result set of a `texdoc` query given the provided keyword, and

`view` to view the resource specified by an entry; it returns a boolean to report whether the process exited successfully.

4 TEX-related tools

The third group in the Island refers to TEX-related tools. We currently hold four projects: TEXprinter, TEXplate, `checkcites` and `arara`.

4.1 TEXprinter

TEXprinter is an application designed for the purpose of printing threads from the TEX community at StackExchange. It can print threads in PDF and TEX formats. The PDF output is provided by the `iText` library. It is a quick option if you do not intend to customize the output. If the thread has images, they are embedded in the final result.

The TEX option is recommended if you want to format the code the way you like. It basically uses the `article` document class, the `listings` package and a fairly straightforward approach. If the thread has images, they are downloaded to the current directory and correctly referenced in the document. Of course, you need to compile it.

TEXprinter ships as a standalone JAR file with dependencies. There is no need to install it, simply download the JAR file from the project repository and execute it. Keep in mind that it needs the Java virtual machine installed.

4.2 TEXplate

The next project in the Island is TEXplate, a tool for creating document structures based on templates. The application name is a word play on *TEX* and *template*, so the purpose may be clear: we want to provide an easy and straightforward framework for reducing the typical code boilerplate when writing TEX documents. Also note that one can easily extrapolate use beyond articles and theses.

The application is powerful enough to generate any text-based structure given that a corresponding template exists. TEXplate can also be used for package writers in generating automated tests. The tool is already available in your favourite TEX distribution.

4.3 checkcites

The third project in the Island is `checkcites`, a Lua script written for the purpose of detecting unused or undefined references from both auxiliary or bibliography files. We use the term *unused reference* to mean a reference present in the bibliography file but not cited in the TEX file. The term *undefined reference* is exactly the opposite, that is, the item cited in the TEX file, but not present in the bibliography file. The script supports two backends, BibTEX and `biber`, and can detect files from your TEX tree.

4.4 arara

Finally, the fourth project in the Island is `arara`, the cool TEX automation tool. This is probably our most well-known and widespread project, as well as the oldest one.

`arara` is a TEX automation tool based on rules and directives. It gives you a way to enhance your TEX experience. The tool is an effort to provide a concise way to automate the daily TEX workflow for users and also package writers. Users might write their own rules when the provided ones do not suffice. `arara` is currently at version 5.1.3; some noteworthy features of version 5 include:

- Support for working directory: users may now specify their working directory by specifying a command line option, so commands will run from a different directory than the directory `arara` launched in. This is especially useful when calling a TEX engine as they resolve files against the working directory.
- Support for processing multiple files: `arara` is able to compile multiple files at once by providing multiple files as arguments. Please note that they should reside in the same working directory. Every other kind of compilation of multiple files is restricted by the mechanisms of the running programs.
- Updated rule pack: `arara` features more than 60 rules ready for use, including TEX engines (with support for both stable and development branches) and assorted tools for graphics, bibliography, indexing, cleaning and conversion between file formats. A typical user might rely just on this pack, without the need of writing a custom rule.

Version 5.1 of `arara` is already available in TEX Live 2020. Simply execute `arara` in your terminal and have fun. The Island has an interesting workflow for `arara`. When a new version is ready to be released, we simply tag it and GitLab will build a CTAN release. Our build script compiles and packages the

application binary, typesets the documentation using our Docker images, assembles the CTAN tree, as well as the accompanying TDS ZIP, and provides a full CTAN ZIP file artifact ready for download. Then we simply send this file to our friends at CTAN.

We are discussing some new features and development paths for the next major version of `arara`.

- We plan to split our tool into an API, a core implementation (that is, a library) and the implementation of the executable (that is, a command line interface). Projects relying on code in the `arara` JAR distributions have to be updated (which might be a potentially breaking change).
- Languages will have to be passed as IETF BCP 47 codes. The old system will be removed.
- Localization will be provided by classes as a library instead of property files in `arara`'s resources. We want to decouple localization from the core implementation and make it easy for users to contribute their own messages.
- The log file shall be specified as path anywhere on the file system. This behaviour, however, may be altered for a future safe mode.
- We are working on a Kotlin DSL (domain specific language) to gradually supersede the YAML-based rule format. The new format aims at being significantly more expressive, easier to write and debug, and less error-prone.
- Elements marked as deprecated in version 5.0 are effectively removed, such as the `<arara>` shorthand notation. This will be a breaking change.
- We are working on a project configuration file format based on a Kotlin DSL as a means to provide resource dependency management, enhanced automation and several additional features. As a consequence, the `preamble` command line option and corresponding configuration map entry will be replaced.

You can follow the progress of version 6.0 in our repository, under the development branch. You can also join our Gitter chat room to learn more about our plans for this major milestone. And of course you are invited to contribute.

5 Editors

The fourth and last group in the Island refers to TEX editors. We currently hold one project which is an application named ArTEXmis.

ArTEXmis aims at being an opinionated, powerful TEX editor designed for all users, especially for package writers and kernel developers. We want to provide an elaborate, immersive user experience when writing TEX code, with a comprehensive and extensive set of features and actions. This project is still under development and we have not provided any public releases so far. We are working on it, so stay tuned for updates.

6 Final remarks

The Island of TEX is hosted at GitLab. Some projects originally hosted at GitHub, such as `arara` and `checkcites`, are now simply mirrored, so the development is consolidated under our organization. Issues and merge requests are welcome.

The TEX ecosystem is a very challenging yet exciting place. We have plans to migrate more tools to the organization in an organic way. At the moment, we have a new project being discussed with the core team. You can learn more about us from our official GitLab repository as well as *TUGboat* articles and electronic mail.

◇ Island of TEX (developers)
`https://gitlab.com/islandoftex`

The design concept for llmk — Light LaTeX Make

Takuto Asakura

Abstract

It is a matter of joy that we have many options for processing a LaTeX document. We can choose the most suitable TeX engine and external programs, such as for bibliography and for indexing, depending on one's needs. However, now it is hard or even impossible in some cases to know what is the 'right' workflow to process a document only by seeing document sources.

Light LaTeX Make (llmk) is yet another build tool specific for LaTeX documents, intended to remove such ambiguity in the workflows. Its aim is to provide a simple way to write down a workflow for document authors and encourage people to always explicitly show the right workflow for each document. For this goal, the design of llmk gives primary consideration to convenience and portability. For example, it supports multiple magic comment formats to enable users to easily write the workflows and it requires only `texlua`, so that it will work under any environment which has LuaTeX.

1 The goal and design concept

TeX, LaTeX, and their friends have a long history and a variety of related software has been developed, including variations of TeX engines, DVIware, and supporting programs such as BibTeX, MakeIndex, and their alternatives. Thanks to such a rich ecosystem, we have numerous options for TeX workflow to create a document. However, on the other hand, there are so many possible *workflows* for processing a LaTeX document, and therefore it is not necessarily easy to detect the right workflow only from the document sources. In addition, there is no ultimate general workflow that can be used for every purpose. Using pdfTeX is one of the typical choices for creating a document in English, but in some cases, it is reasonable to choose XeTeX or LuaTeX, e.g., if you want to use fonts installed in your system independent of TeX systems. For these reasons, LaTeX users should clearly specify the workflow for each document, at least for those documents where the sources will be seen by someone else.

There are a number of existing well-established generic build tools, such as (GNU) Make, that can be used to explicitly specify the workflows. However, for many simple LaTeX documents, such as those that require only a single pdfTeX run, it might be rather overkill to utilize such tools. As a matter of fact, people often neglect using them for small documents and leave the right workflows as mysteries. Also, it is difficult to regard knowledge of such generic build tools as an essential skill for all LaTeX users, especially considering light users and non-programmers.

Focusing on such casual use cases, I began a new project named "Light LaTeX Make" (llmk). Its goal is to encourage people to always explicitly show the workflow for each document by providing convenient ways to do it. The design of the tool is all about this purpose. First, it supports multiple magic comment formats to specify the workflows in addition to external configuration files. Magic comments are an easier way than external files, though the difference is small. It should be compatible with most LaTeX use cases, including using it on cloud services and LaTeX-specific IDEs.

Second, it is fully cross-platform. It requires only `texlua`, and thus it should work in almost all TeX environments. For instance, one does not need to install any dependency other than the TeX Live distribution.

Third, it behaves exactly the same in any environment. At this moment, llmk intentionally does not provide any method for user configuration, so that a LaTeX document with a supported workflow specification should be processed exactly in the same way, no matter where you run the program.

Overall, llmk is a tool to provide a convenient way to describe the workflow for an individual LaTeX document. It is designed more or less for simple documents and might not be suitable for large projects that require complicated workflows. For such cases, more sophisticated tools are better suited. A well-written document that already has a `Makefile` or similar is not the target of this project. In such a document, the right workflow is already explicitly shown. The major targets of llmk are small documents without unusual requirements.

There are other LaTeX-specific build tools with aims similar to llmk. The differences from such tools will be discussed later (Section 3).

2 How llmk works

In this section, only a brief summary of the usage and the mechanism in llmk is given. The details are shown in the bundled documentation.

2.1 How to write the workflows

A user of llmk can write a document's workflow in a special external file (`llmk.toml`) or in the TeX file (`*.tex`) itself. When the `llmk` command is executed without any argument, it loads the `llmk.toml` file in the working directory. If one or more names of TeX

files are specified as arguments for `llmk`, it reads the *TOML fields* in the files — these are special comment areas that are given by comment lines containing only three or more consecutive `+` characters:

```
% +++
% latex = "xelatex"
% +++
\documentclass{article}
```

Either way, you can write the workflow in the TOML format [5] — a small configuration-oriented language. This language is designed to be human-friendly and is used in numerous projects.[1]

General-purpose programming languages, such as Perl and Lua, can also be used for writing workflows and are in fact used in some TEX-related build tools, but they are too powerful and have large specifications. Smaller languages designed specifically for configuration, which are easier to learn, are better for llmk. Among the various configuration languages, including JSON and YAML, TOML is easy to parse and thus a built-in parser can be written in reasonable lines of code in pure Lua. These are the reasons why TOML was chosen for llmk.

2.2 Simple keys

There are only a few important keys for llmk configuration for casual users. For simple documents where the default configuration is applicable, using some of these keys should be enough:

`latex` (string) specifies the LATEX command to use. The default value is `"lualatex"`. Since llmk runs on `texlua`, the installation of LuaTEX is guaranteed. This is the reason that LuaTEX is chosen for the default engine. Similar keys `dvipdf`, `bibtex`, etc., are also available.

`max_repeat` (integer) sets the maximum number of repetitions. For various reasons, such as solving cross-references, llmk has a feature to repeat command executions. This key exists to prevent potential infinite loops. The default value is `5`.

`source` (string or array of strings) sets the source TEX files to process. This key is effective, and required, only in `llmk.toml`.

The following is a small example of a configuration for llmk which overrides the defaults:

```
# source TeX files
source = [ "test1.tex", "test2.tex" ]
# software to use
latex = "xelatex"
bibtex = "biber"
# misc
max_repeat = 7
```

When a value of a wrong type is given for a key, it will result in a type error *before* llmk tries actual document processing. It is designed to produce helpful error messages as much as possible, not to add confusing errors in addition to those produced by TEX engines.

2.3 Flexible control

For most simple LATEX documents, just using simple keys described in the previous section should work fine. Though such documents are the main targets of llmk, it has features to process more complicated documents if users desire to do so.

The core of flexible control in llmk is a pair of keys: `sequence` (array of strings) and `programs` (table of tables). The `sequence` array holds the names of programs in the order of execution, and the `programs` table contains detailed configuration for each program in the sequence.

The default configuration of llmk is designed to work without changes for typical LATEX documents. Users are required to write only the differences from the default, so that they do not have to write all configurations from scratch every time. The default value of the `sequence` array is as follows:

```
["latex", "bibtex",
 "makeindex", "dvipdf"]
```

Under this configuration, llmk tries to convert `*.tex` files to `*.pdf`. In case `*.dvi` is generated in the process, the `dvipdf` program (by default DVIPDFMx) is executed to convert it to a PDF. The `bibtex` and `makeindex` programs are executed only if the corresponding files (`*.bib` and `*.idx` respectively) exist, and the `latex` program is set as `postprocess` in order to make sure to rerun the LATEX command after those executions.

2.4 Supports for other formats

For the convenience of the users, llmk supports other existing magic comment formats. At present, the so-called shebang-like magic comment, which is supported by a few existing tools, notably the YaTeX mode for Emacs,[2] is supported by llmk. Writing `%#!pdflatex` in the first line of a `*.tex` file is equivalent to specifying `"pdflatex"` to the `latex` key. Other formats are also planned to be supported.

3 Differences from other tools

In response to the most frequently asked question, I will briefly explain the differences from other similar

[1] You can find the list of projects using TOML in its official wiki: `https://github.com/toml-lang/toml/wiki`.

[2] `https://www.yatex.org/`

LaTeX-specific build tools. Please note that most of these differences are just the result of different design concepts, and I would *not* call them 'advantages'. Though the aims and concepts that each tool prioritizes are a bit different from each other, they all have longer histories than llmk and thus have sophisticated designs and implementation. I have been greatly inspired from them and will continue to learn. I hope llmk can provide another useful option for LaTeX users and some new ideas and inspiration for the developers.

3.1 Latexmk and rubber

Latexmk [2] and rubber [3] are two well-known LaTeX-specific build tools. They have their own characteristics and have stable sophisticated implementations, but their purposes are slightly different from that of llmk. Their goals are to provide easy ways to process LaTeX documents; they guess how to process a document by analyzing the log files, for instance, and implicitly determine the process. In other words, they try hard to 'hide' the specific workflow from users as much as possible. In addition, for both tools, users are allowed to choose some variations, e.g., a favorite TeX engine from pdfTeX, XeTeX, LuaTeX, etc., with the command-line options. It is a useful feature, but this makes it harder to reproduce the same process for colleagues without being told another piece of information, i.e., runtime command-line options, from authors of documents.

On the other hand, llmk takes a different approach: it requires users to explicitly show the workflow to process a document either in an external configuration file (`llmk.toml`) or in a `*.tex` file. Thanks to its default configuration, it appears as if llmk determines the workflow automatically for simple configuration, often consisting of a single `latex` key, but in fact this is just a 'shorthand' for one of the typical workflows and nothing is implicitly determined. Thus, once you want to process a more complex document for which the default configuration is unsuitable, llmk will require you to specify everything explicitly. In this way, we can take advantage of both convenience and portability.

3.2 Arara and spix

Arara [1] is a newer build automation tool for LaTeX documents that has become quite popular. Its aim is close to ours: arara provides a way to describe the workflow explicitly for each document. It has a set of *rules* indicating the ways to process typical LaTeX documents and a user can specify which rules with a *directive*, which is a magic comment in the `*.tex` file. It also enables users to create their own rules by writing the details in external files, in case a suitable built-in rule is missing. Arara is a big project and is capable of processing large documents that need complicated workflows, while llmk is small and more or less focusing on simple documents.

Spix [4] identifies itself as a simpler version of arara. It also follows the idea of explicit workflow description for each document and generally focuses on simple documents. Therefore, the goals of spix and llmk are almost the same, though there are a few differences in concrete syntaxes and specifications.

One apparent strength of llmk as compared to these two tools is that llmk can be executed without installing any dependency other than from TeX systems. While arara and spix are implemented in Java and Python respectively and thus require external programs in order to use them,[3] llmk is written in pure Lua and thus can work with only `texlua` available. The specification and features of llmk are far smaller than those of arara. Instead, llmk prioritizes a uniform way to describe the workflows available for nearly all TeX environments.

4 Acknowledgements

This project has been supported by the TeX Development Fund created by the TeX Users Group (No. 29). I would like to thank all contributors and the people who gave me advice and suggestions for new features for the llmk project. I am grateful to Yusuke Kuroki for helping with the manuscript.

References

[1] Paulo Cereda, et al. *arara — The cool TeX automation tool.* `https://ctan.org/pkg/arara`

[2] John Collins. *latexmk — generate LaTeX document.* `https://ctan.org/pkg/latexmk`

[3] Sebastian Kapfer. *rubber — a building system for LaTeX documents.* `https://launchpad.net/rubber/`

[4] Louis Paternault. *SpiX — Yet another TeX compilation tool: simple, human readable, no option, no magic.* `https://ctan.org/pkg/spix`

[5] Tom Preston-Werner. *TOML: Tom's Obvious Minimal Language.* `https://toml.io/`

◇ Takuto Asakura
The University of Tokyo
Department of Computer Science
`tkt.asakura (at) gmail dot com`

[3] Neither the Java virtual machine nor the Python interpreter are included in TeX Live, or in MiKTeX.

Typesetting product catalogs and other database-driven documents with the speedata Publisher

Patrick Gundlach

Abstract

The speedata Publisher is a database publishing system based on LuaTEX. Although it is a fully commercial-driven development, it is available under an open source license (AGPL). The main goal of the speedata Publisher is to provide the high quality of TEX's typesetting output while fulfilling the needs of database publishing. The speedata Publisher implements its own language for defining layout rules. This language is inspired by HTML, XSL and CSS, and specializes in layout generation and excels in optimizing layout such as rearranging objects on the page, whitespace optimization, copyfitting and other means.

LuaTEX has the ability to manipulate and build the internal data structure that TEX uses to assemble the pages and to break paragraph into lines. It provides an extremely powerful environment for non-standard typesetting tasks by allowing all necessary steps to be done programmatically while still falling back to TEX's algorithms.

1 Introduction

LATEX is great if you want to write articles or books. But how about if you want to publish product catalogs or data sheets? Is TEX still the tool of choice?

Normal text vs. data from databases

Product catalogs and similar documents are at best created from data stored in a database. But what distinguishes normal text from data retrieved from databases?

Texts in a TEX document could look like this

```
The quick\footnote{yes, really quick} brown
fox jumps...
```

while data from the database has a rather schematic structure:

```
<productdata>
  <articlegroup
        name="interior lights"
        number="123">
    <article number="123-12345">
      <property1>...</property1>
      <property2>...</property2>
    </article>
    <article number="123-12346">
      <property1>...</property1>
      <property2>...</property2>
    </article>
  </articlegroup>
</productdata>
```

Data processed in such systems is almost exclusively formulated in XML, regardless of how it is stored on disk. Textual descriptions are generally stored either as plain text or as HTML formatted text, rarely as Markdown formatted text.

A fundamentally different approach is therefore necessary to process data from databases. The processing of documents is no longer linear, i.e. from 'top to bottom'. Instead, the data must be assembled according to a logic that differs from application to application. From the data above, it is not possible to see how it should be represented. Whether it is intended to be typeset as a table, a nicely designed page with several products, or as a data sheet — this cannot be seen from the data alone.

1.1 Strict separation of data and layout

Apart from a few exceptions, no information about the appearance is found in the data. While the strict separation is in theory a good concept, it makes the typesetting part much harder. Sometimes the separation is even impossible to maintain. For example: when having a full page background image and text to be placed in a *good* position, you need some information coming from a human decision.

So how do you arrange the data on a page? According to which rules must the elements be placed? To get to the solution, it helps to look over the shoulder of a graphic designer when creating a document (e.g., a product catalogue). Professional graphic designers work according to rules: How is a catalog structured? Which products should be displayed in which way? How many products fit on one page? Which colors and fonts are used? Where is a page break inserted?

The same rules are usually applied when filling the pages, even if pages often look very different. If you can manage to write these often formal rules in text form or in a programming language, you are often already very close to the goal.

1.2 Software used for product catalogs

Adobe InDesign is certainly the software most often used to create documents with non-linear layout. It is professional desktop publishing software for Windows and Mac. This very powerful program has excellent graphic qualities and can be automated by plugins. There is also QuarkXPress, which works similarly. However, these programs have limitations in automation. In practice, these programs usually work by using a database interface and page templates to fill pages. When finished, the pages need

to be finalized via a tedious manual process. This workflow is suitable for documents that change only occasionally, but quickly reaches its limits if the database changes very frequently or if documents are to be generated fully automatically.

1.3 TEX as an alternative?

For the readers of *TUGboat*, the question is of course how far TEX can be used here. The advantages of TEX are well-known to us:

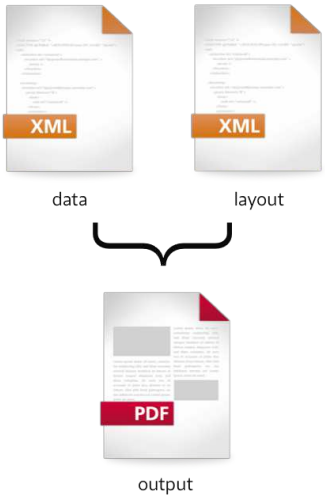
- Full automation. You can set up the process so that a document is generated at the push of a button or at a given time.
- Free software. No dependence on proprietary software. TEX can be used on any number of computers without any additional license fees.
- High output quality. We all know that TEX's line breaking algorithm is superb.
- High speed. TEX can output up to 300 pages per second on my old 2015 laptop with almost no startup time. (Less than that with more complex documents and lots of fonts.)

A few difficulties remain, however:

- XML encoded in UTF-8 as input format. TEX is neither made for XML input nor for UTF-8 processing. LuaTEX allows more than 256 glyphs in a font, so that helps significantly. Since April 2018 LATEX defaults to UTF-8 input, so this part is getting better.
- Assembling the data. As seen, the input is not linear. You have to go back and forth within the data and fetch data from different XML elements in the input.
- Output of HTML sources. Text in database publishing is usually stored as plain text or as HTML fragments. To render HTML, a CSS+HTML parser is needed.
- Optimization of pages. Many applications demand some kind of whitespace optimization such as adding images to the page until the text completely occupies the space. Other applications require, for example, reducing the text size until the text fits on one page (copyfitting).

2 speedata Publisher

I have developed the speedata Publisher precisely for the purpose of non-linear documents with high demands for layout flexibility. It is open source software based on LuaTEX. You can download it from the homepage [1] and use it immediately without any further installation of dependencies (not even TEX is required; it is included in the ZIP file). A comprehensive manual [2] describes in detail how to use the software.



In addition to the data, which must be available in XML format, the layout instructions are also formulated in XML. This has several advantages.

- You stay in the XML environment, which is required for handling the data anyway.
- With a schema, editing XML in a text editor is also fun.
- You see syntax errors immediately.
- XML can be easily created and transformed from programs.

2.1 The speedata layout language

Since the data can be structured in any way, the layout language must be very flexible. It must also be possible in the layout files to formulate and evaluate the above-mentioned design rules. Supporting queries to the data is necessary, such as: how wide has the object become? Is there still enough space on the page?

Existing layout or formatting languages do not allow such flexibility. (X)HTML has no programming support, XSL has no knowledge of layout, CSS is fine for output, but has only limited programming abilities. XSL-FO is rigid in its output and has no way to respond to dynamic queries. In this respect, the layout language is a mixture between the languages mentioned here.

2.2 Hello, world

In the following sections I would like to give a small insight into the layout language. The classic "Hello, world" example serves as an introduction to speedata.

In database publishing the input usually consists of two files: the data file and the layout file. I ignore images and font files for now.

The data file in the "Hello, world" example consists of one line:

```
<greeting>Hello, world!</greeting>
```

This file must be saved in an otherwise empty directory under the name `data.xml`.

The layout file (`layout.xml`) is rather larger, and looks scarier than it really is (the arrow indicates an editorial line break):

```
<Layout
  xmlns="urn:speedata.de:2009/publisher/en"
  xmlns:sd="urn:speedata:2009/publisher/  ↙
    functions/en">
  <Record element="greeting">
    <PlaceObject>
      <Textblock>
        <Paragraph>
          <Value select="."/>
        </Paragraph>
      </Textblock>
    </PlaceObject>
  </Record>
</Layout>
```

If you have the speedata Publisher installed, you can run the command `sp` to create a PDF file.

The processing starts at the root node in the data file. In this case it is the element `<greeting>`. It sets the "focus" to this element so you can access it from the layout file. The software now looks for an entry point in the layout file (`<Record>`) and executes every command that is found within this `<Record>` element. In this example, a text block is output for the current element. The `<Value>` uses the dot (`.`) to select the contents of the current focussed element in the data file, in this case just the text "Hello, world". The dot is a so-called XPath expression with which you can select data. A more detailed description of the "Hello, world" example can be found in the manual [3].

2.3 Dynamic layouts

To enable dynamic layouts, the speedata layout language has programming options such as loops and variables and the ability to query the appearance of the page and objects that are put in virtual areas. Together, these have enough expressiveness to implement complex layout requirements. As with TEX, you can put any objects into a virtual area which is not output in the PDF. In TEX this is called a box; here it is called a group. For example, to compare the width of a group to the size of the page and act on the outcome of the comparison:

```
<Switch>
  <Case test="sd:group-width('mybox') >   ↙
    sd:number-of-columns()">
    <!-- too wide, recalculate -->
  </Case>
  <Otherwise>
    <!-- great, fits on the page -->
    <PlaceObject groupname="mybox" />
  </Otherwise>
</Switch>
```

The requirements in practice are of course much more complex. Interestingly, however, a few building blocks are sufficient to create complex layouts. Many functions in speedata Publisher fall into the category *syntactic sugar*, i.e. not strictly necessary but helpful constructions. For example, an image file can be specified as a fallback for image output:

```
<Image file="myfile.pdf"
       fallback="placeholder.pdf" />
```

This could also be written in a different way:

```
<SetVariable
      variable="filename"
      select="'myfile.pdf'" />
<Switch>
  <Case test="sd:file-exists($filename)">
    <Image file="{ $filename }"/>
  </Case>
  <Otherwise>
    <Image file="placeholder.pdf"/>
  </Otherwise>
</Switch>
```

2.4 Grid typesetting

Objects can be placed anywhere on a page or in a grid. Grids can be any size and define a coordinate system that helps in placing objects automatically and ensuring that no object overlaps any other.

```
<Layout xmlns="urn:speedata.de:2009/  ↙
    publisher/en"
  xmlns:sd="urn:speedata:2009/publisher/  ↙
    functions/en">

  <SetGrid height="12pt" nx="10"/>
  <Trace grid="yes"/>
  <Pageformat width="8cm" height="4cm"/>

  <Record element="data">
    <PlaceObject column="3" row="2">
      <Textblock>
        <Paragraph>
          <Value>Hello world!</Value>
        </Paragraph>
      </Textblock>
    </PlaceObject>
  </Record>
</Layout>
```

Using a grid has several advantages:

- Every object allocates an area on the page. It is easy to check how big this area is.
- Objects that are placed on a grid cannot overlap, unless forced to. The system moves the object to the next free space.
- It is easy to achieve typesetting on a grid just by letting the output start at a new grid row.

Of course not everything can be placed within a grid. Logos or background images for example need to be placed at absolute positions:

```
<!-- grid -->
<PlaceObject row="4" column="5">
    <Image file="_samplea.pdf" width="5"/>
</PlaceObject>

<!-- absolute -->
<PlaceObject row="12mm" column="5cm">
    <Image file="_samplea.pdf" width="5"/>
</PlaceObject>
```

2.5 Other features

The speedata Publisher has many, many features. Here, I'd like to highlight just a few of them.

Accessibility It is possible to attach logical structure to the texts placed in the PDF so it can be PDF/UA (Universal Accessibility) compliant.

HTML input The speedata Publisher comes with a CSS+HTML parser that lets you typeset documents written in HTML as they would look in a browser.[1]

Master pages Page templates, including logos and other static and dynamic information, can be defined together with arbitrarily complex conditions for when the page will be chosen by the software.

Page areas You can define areas on the page to let text flow from one area to the next area. This is used in magazine typesetting.

HTTP assets Images and all other resources can be loaded on the fly from the web. This makes it easy to use digital asset management software.

[1] This feature is under development, so not all aspects are implemented yet.

Image wrapping Images can be (automatically) enriched with information about where text can wrap around the image. The paragraph shape is calculated automatically.

Advanced tables The speedata Publisher does not use any of TEX's table code. It ships with its own table model, which is inspired by HTML and supports static and dynamic headers and footers, controllable page breaks, running totals, complex table cell backgrounds and much more.

Server mode Included in the Publisher is a REST API that listens for incoming HTTP requests to start publishing runs. This makes it easy to build a server infrastructure for typesetting jobs.

Strong quality assurance There are more than a hundred documents that are automatically compared before making changes to the software, so we can be assured old documents will be typeset without changes in future versions.

3 speedata and LuaTEX

As mentioned above, LuaTEX is used as the backend for speedata. Almost all parts of the speedata Publisher are written in Lua. No code from the plain, ConTEXt or LATEX formats is used. There is a tiny TEX wrapper that jumps directly into the Lua mode, which does all the processing.

```
\catcode`\{=1 \catcode`\}=2
\directlua{require("publisher.spinit")}
\end
```

All other functions are at the Lua level. These are, for example

- Parse the XML files (data and layout)
- Read in all images and font files
- Execute the program statements in the layout
- Assemble the data structures for TEX
- ... and much more

Some of the routines are written in the programming language Go and included as a library at runtime. This library handles the loading of resources via HTTP (including caching) and parsing of HTML and CSS files. It was easier to use existing libraries for these tasks than to rewrite them in Lua from scratch.

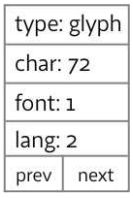
3.1 TEX without \TEX

If no input comes in the form of TEX code, how is LuaTEX able to typeset text and place other objects into the PDF?

TEX normally reads the files with the macro instructions (e.g., `\section`) and stores the contents as so-called nodes after some processing. These are the smallest data units, which store e.g. a character or a

glue. With these data units everything that is visible in the output (along with some other technical information) is represented. These data structures can then be used to create DVI or PDF output. Thanks to LuaTEX, these nodes can also be created and manipulated in Lua. Thus, the main part of the Lua program code consists of generating such nodes from the input data and the instructions of the layout file.

Node lists are linked lists of single nodes, which can also contain lists themselves. For example, the content of a horizontal box `\hbox{...}` is a list and the box itself can be part of another list. Each node consists of different fields, depending on what is stored. For example, the character "H" could be represented as a node as follows.

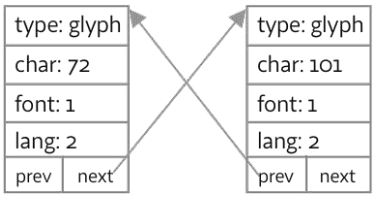


Such a character could easily be created with the following Lua commands (the double dash `--` is a Lua comment):

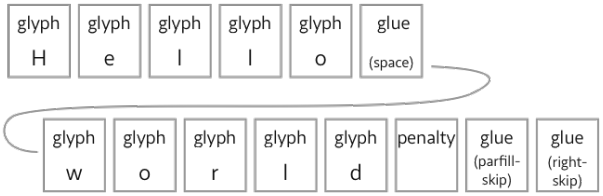
```
h = node.new("glyph")
-- 72 is the ascii code for H
h.char = 72
h.font = 1
h.lang = 2
```

You can chain the nodes by, for instance, setting the `prev` and `next` pointers:

```
-- as above
e = node.new("glyph")
e.char = 101; e.font = 1; e.lang = 2
h.next = e
e.prev = h
```



In this way, entire nodelists can be generated.



We are close to a nodelist that can be used for output. Three things are missing from a "perfect" paragraph:

1. hyphenation
2. kerns
3. ligatures

Hyphenation: there is a Lua function for TEX's hyphenation routine: `long.hyphenate(nodelist)`. When called, LuaTEX changes the nodelist and inserts the so-called discretionaries that signal a hyphenation point.

Kerns and ligatures: there are two very helpful routines that add ligatures and kerns to the nodelists: `node.ligaturing(nodelist)` for ligatures, and for kerns `node.kern(nodelist)`. The former replace some glyph nodes with ligatures, so that they can be dissolved again when a word is hyphenated. The latter inserts small (often negative) spaces between glyphs. Regarding ligatures, one can argue whether it is still appropriate to do this via the TEX mechanism. OpenType fonts often contain other ligatures that would have to be translated for TEX's ligature mechanism. Furthermore, libraries like HarfBuzz offer much more powerful functions for ligatures.

If hyphenation, kerns and ligatures are inserted, you can use

```
tex.linebereak(nodelist, parameter)
```

to create a paragraph broken by TEX. The parameters specify the values for paragraph settings like emergencystretch or linepenalty but also the paragraph style (parshape). The result is a vertical box with single lines in horizontal boxes.

3.2 Output of nodelists

After nodelists have been assembled, they can be output. The speedata Publisher collects all material for the pages and outputs it in one go. `tex.shipout(n)` outputs the TEX box with the number n:

```
nodelist = node.vpack(nodelist)
tex.box[1234] = nodelist
tex.shipout(1234)
```

Before output, structural elements may have to be written to the PDF for PDF/UA. To do this, the content of the page is analyzed and a PDF object structure is written to the PDF for accessibility purposes.

3.3 Fonts and languages

In the example above, we just used some dummy values for 'font' and 'lang'. Usually TEX loads the font files or language patterns with `\font` and `\patterns`. The speedata Publisher has its own routines for both to allow UTF-8 input, similar to `fontspec` and `luaotfload` for LATEX.

A new language can easily be loaded in LuaTEX:

```
local l = lang.new()
l:patterns(pattern)
local id = l:id()
```

Here `pattern` is the content of a pattern file.

Loading a new font is a bit more complicated. The FontForge library that is part of LuaTEX is used to get information about (OpenType) fonts. An alternative routine based on HarfBuzz is planned, which is part of LuaTEX (under the name `luahbtex`) since the last TEX Live release.

```
font, err = fontloader.open(filename_with_path)
...
fonttable = fontloader.to_table(font)
```

`fonttable` now has all font properties in an extensive table, which can be made available to TEX:

```
local f = { }
f.name          = fonttable.fontname
f.fullname      = fonttable.fontname
f.designsize    = size
f.size          = size
f.direction     = 0
f.filename      = fonttable.filename_with_path
f.type          = 'real'
...
f.characters = {
    -- code for all glyphs in a font
}

-- define the font:
fontid = font.define(f)
```

You can use the font id in the nodelists above.

3.4 PDF specials

The PDF contains a lot more information than that which is visible at a first glance. For example, bookmarks, hyperlinks, document structure for accessibility, attached documents for electronic invoices are elements that need to be written into the PDF. Thanks to pdfTEX and the API in LuaTEX, this is an easy task once the required syntax for these PDF objects are known. They can be written to the PDF as follows:

```
pdf.obj(...)
```

This function has several different parameters that allow compressed or uncompressed text or data to be written as a simple or a stream object.

There are also visible objects that cannot be created with TEX's graphics routines. Colors, transparency, shades and other objects need to be written with PDF drawing instructions. For example, the borders in the following picture need to drawn with instructions such as `0 0 m 1 5 l` which means "move to position (0,0) and draw a line to (1,5)". There are operators to draw lines and Bézier curves, fill paths, clip contents from inside and outside of given areas and many other drawing operators.

These operations can be inserted into the PDF by *whatsits*:

```
n = node.new("whatsit","pdf_literal")
n.data = "0 0 m 1 5 l"
```

and then insert this whatsit into the nodelist (output grayscaled for *TUGboat*):

4 Outlook and conclusion

Of course I can only scratch the surface in this article. LuaTEX and also the speedata Publisher are two very powerful pieces of software. As can be seen, the speedata Publisher would not be possible in this way without LuaTEX. There is no need to understand TEX's macro language to use TEX, even for the programmer.

The speedata Publisher is in active development since 2009. I have a lot of plans for the future development (such as HarfBuzz integration), but the (paying) customers are the ones who drive most development of new features.

I'd like to invite you to try out the software, ask questions, look at the showcase on the homepage or just browse the manual for inspiration.

To close with Donald E. Knuth's words: Go forth now and create masterpieces of the publishing art!

References

[1] speedata homepage. `https://www.speedata.de`.
[2] speedata manual. `https://doc.speedata.de`.
[3] "Hello, world" example in speedata manual. `https://doc.speedata.de/publisher/en/helloworld/`.

◇ Patrick Gundlach
speedata GmbH
Odilostraße 43
13467 Berlin
Germany
`gundlach (at) speedata dot de`
`https://www.speedata.de/`

A first set of LaTeX packages

Jim Hefferon

Abstract

This describes a curated list of packages that covers most of what beginners want to do. It seeks to name one package in each area that is capable and reliable.

1 Overview

At TUG 2019 I reported on using social media to help understand the needs of today's beginners [1]. Often they just need a pointer to the right package. This describes a package list suited to those users.

A list that is exhaustive wouldn't help here; I have kept the document to two sides of a page. Of course that involved making choices. I am sorry that this leaves off some first-quality work, but in any event, the packages named are capable and basically bug-free.

Beyond solving problems, the criteria for inclusion in the list is that a package should be in the distributions and popular. I also value documentation, particularly if it has helpful examples.

Part of the reason for this article is to solicit feedback. I have already made improvements in response to comments on a draft from social media, in [2]. The end product will be a document in PDF, HTML, and video. The PDF will be on CTAN.

Below I will go over the choices. The document core consists of a few sections classifying areas, intended to help a user find packages, which is reproduced below. Each package name is a hyperlink, with a terse description. (There are a few extra comments in parentheses that come up in conjunction with the recommendations.)

Before those is an introduction. It mentions CTAN,[1] the target of almost all the links here. It also mentions using `texdoc` to read local documentation. Finally, it notes that if a person is writing for a journal or institution then they should ask if there is a house package.

2 Every document

To change page size, margins, and orientation, use `geometry`.[2] Get multiple columns with `multicol`.[3]

Any document containing significant amounts of mathematics should use the American Mathematical Society's packages `amsmath`[4] and `amssymb`.[5] I also use `amsthm`[6] for producing theorem environments. Notes: (1) `amssymb` inputs `amsfonts` so you don't need to load the latter, (2) load `amsthm` after `amsmath`, (3) don't load `amsmath` directly, instead get it by loading `mathtools`,[7] which adds some useful improvements.

You can toss in `microtype`.[8] My eye can't spot the improvements but I appreciate that often when I use it, fewer lines need rewriting for overfull boxes.

3 Inside a document

To tweak lists, use `enumitem`.[9]

Enhance captions with `caption`[10] and control floating environments with `float`.[11] (In particular, if you want an option that overrides automatic float placement and puts something exactly where you ask, this package provides the option '`H`'.)

Get hyperlinks and turn references into links with `hyperref`[12] (this should be the last or next to last package loaded). Make cross-references say 'Theorem 1.2' instead of just '1.2' with `cleveref`;[13] load this after `hyperref`. Have urls and file paths that can linebreak with `url`[14] (but `hyperref` has its own facility, so if you are using `hyperref` then omit `url`).

I do code listings with `listings`[15] (although `minted`[16] also has a lot going for it). Make single quotes inside verbatim text come out correctly with `upquote`.[17]

(A tangent: copy and paste for computer code listings would be especially convenient. This is a start for the `listings` package.

```
\lstset{basicstyle=\ttfamily,keepspaces=true,
        columns=fullflexible}
```

But it is not a full solution. For one thing, the result depends on the PDF viewer. Worse, it loses initial spaces in a line — if your code line begins with four blank spaces then after a copy and paste those spaces are gone.)

For code in Python have a look at `pythontex`,[18] which, besides showing the code listings, also allows you to execute Python and put the results in your output. Do the same for the *Sage* mathematics

[1] ctan.org
[2] ctan.org/pkg/geometry
[3] ctan.org/pkg/multicol
[4] ctan.org/pkg/amsmath
[5] ctan.org/pkg/amssymb
[6] ctan.org/pkg/amsthm
[7] ctan.org/pkg/mathtools
[8] ctan.org/pkg/microtype
[9] ctan.org/pkg/enumitem
[10] ctan.org/pkg/caption
[11] ctan.org/pkg/float
[12] ctan.org/pkg/hyperref
[13] ctan.org/pkg/cleveref
[14] ctan.org/pkg/url
[15] ctan.org/pkg/listings
[16] ctan.org/pkg/minted
[17] ctan.org/pkg/upquote
[18] ctan.org/pkg/pythontex

suite with `sagetex`[19] and similar systems exist for R, Haskell, and Scheme.

There are many packages that add table capabilities such as multirow entries and breaking across pages. I most often use `array`,[20] which lets you define custom column types. For units, use `siunitx`[21] (which also has a table column type for aligning on a decimal point).

To make boxes that are colored or framed, such as boxes for theorems, I use `mdframed`.[22]

Finally, when developing a document I often want some filler text. I use `lipsum`.[23]

4 Graphics and color

To include graphics in files and to do simple manipulation such as resizing, use `graphicx`.[24] Use the JPG format for photos, PNG for other kinds of raster graphics, and PDF for vector graphics. If your graphic is in another format then convert it to one of the three. (Usually you give the file name without the extension, as with `\includegraphics{graph}`.) Include parts of a PDF document with `pdfpages`.[25] Include video or sound using `media9`.[26]

To get colors, use `xcolor`[27] (although the documentation can be hard to make out).

For plots and graphics I use *Asymptote*,[28] a development of METAPOST with three-dimensional features. However, many people instead use Ti*k*Z[29] to draw graphics inside the document.

5 Front and back matter, headers, footers

To style chapter and section titles, use `titlesec`.[30] For page headers and footers, reach for `fancyhdr`.[31] You can tweak the format of tables of contents, lists of figures, etc., with `tocloft`.[32]

Write answers to exercises to an external file so you can read them in later with `answers`.[33] I like footnotes at the page bottom, so I use `footmisc`[34] (but I had to hack to change the space between a footnote mark and the footnote). Make an index with `makeindex`.[35] Bibliographies are a thorny area, with lots of strict requirements. CTAN is a big help here since it has many styles for both BibTeX[36] and BibLaTeX.[37]

6 Special documents

Make exams and problem sets with the `exam`[38] class.

There are many, many resume and CV packages. Have a look at CTAN's *cv* tag.[39]

To make presentations, use the `beamer`[40] class. (But with this package you are entering another world, where many of the packages discussed here do not work. For example, section title styling happens via a completely different mechanism.)

7 Fonts and engines

To see options besides the default Computer Modern fonts, visit the LaTeX Font Catalogue,[41] which includes copy and paste code to make each one work.

Beyond that list, you can also use any font that your computer has (which usually works well only if your document does not have much mathematics). To convert LaTeX source to PDF there are three main programs, called engines. Most people use pdfLaTeX. The XeLaTeX engine and the LuaLaTeX engine can leverage the `fontspec`[42] package to use your system's fonts. (A word about the preprint site `arXiv.org`. If your document was produced with XeLaTeX or LuaLaTeX then you can only submit a PDF, not the document source.)

8 What's missing?

Again, I would be glad to hear suggestions for making this list better.

References

[1] J. Hefferon. What do today's newcomers want? *TUGboat* 40(2):106–108, 2019. `tug.org/TUGboat/tb40-2/tb125heff-newusers.pdf`

[2] Various Reddit users. A First List of LaTeX Packages. `old.reddit.com/r/LaTeX/comments/hpal2i/a_first_list_of_latex_packages/`.

◇ Jim Hefferon
Saint Michael's College
`jhefferon (at) smcvt dot edu`
`https://hefferon.net/`

[19] `ctan.org/pkg/sagetex`
[20] `ctan.org/pkg/array`
[21] `ctan.org/pkg/siunitx`
[22] `ctan.org/pkg/mdframed`
[23] `ctan.org/pkg/lipsum`
[24] `ctan.org/pkg/graphicx`
[25] `ctan.org/pkg/pdfpages`
[26] `ctan.org/pkg/media9`
[27] `ctan.org/pkg/xcolor`
[28] `asymptote.sourceforge.io`
[29] `ctan.org/pkg/pgf`
[30] `ctan.org/pkg/titlesec`
[31] `ctan.org/pkg/fancyhdr`
[32] `ctan.org/pkg/tocloft`
[33] `ctan.org/pkg/answers`
[34] `ctan.org/pkg/footmisc`
[35] `ctan.org/pkg/makeindex`
[36] `ctan.org/topic/bibtex-sty`
[37] `ctan.org/topic/biblatex`
[38] `ctan.org/pkg/exam`
[39] `ctan.org/topic/cv`
[40] `ctan.org/pkg/beamer`
[41] `tug.org/FontCatalogue`
[42] `ctan.org/pkg/fontspec`

Presenting our LaTeX workshop online

Susan DeMeritt, Cheryl Ponchin

Preparing for our first-ever video LaTeX class was much harder than we thought it would be. We would each start a video, find an error or make an error while speaking and have to start all over again. This would happen several times in a row. It was very frustrating.

Once we finally completed the videos and were happy with them, we sent them to Paulo Ney de Souza for processing and posting.

One of the things we noticed, comparing the video class and in-person class, is that if we make a mistake in the in-person class we can recover much more easily, while in the video class we would have to start over.

It was interesting to watch the videos as they were shown on Thursday, July 23rd, before the conference. We liked being able to answer questions via chat. We didn't have to stop the videos to answer questions. There were also people adding various pieces of information that was good for all of the attendees to learn.

Overall, we thought it went really well. Definitely a good option since no one could attend the conference in person because of the pandemic. Paulo Ney de Souza was so helpful getting everything set up for us. Cheryl sent him all of the videos and Paulo put them in the order we requested.

The table of contents for the syllabus is included here. The full syllabus, both the LaTeX source and output PDF, is linked from `https://tug.org/tug2020/workshop.html`.

◇ Susan DeMeritt
Institute for Defense Analyses
`sue (at) ccrwest dot org`

◇ Cheryl Ponchin
Institute for Defense Analyses
`cheryl (at) idaccr dot org`

LaTeX Workshop
TUG 2020 Conference

learnlatex.org: Taking LaTeX training fully interactive

David Carlisle, Paulo Roberto Massa Cereda, Joseph Wright

1 Introduction

There are a plethora of resources available to new LaTeX users. However, it is much more difficult to discover which of these provides the best introduction to LaTeX. These online resources vary in quality and correctness: over time, and with limited editing, even good advice can become out-of-date. Many good resources are over-detailed for a new user who needs only straightforward help to get over the initial barrier to using the system.

For many programming languages there are now web sites providing the opportunity to try coding online using a cloud compiler. These cloud compilers can be harnessed by a range of teaching websites to offer a simple introduction to the language using a suitable IDE (integrated development environment); a good example is `LearnPython.org`. Such sites tend to be limited in scope as they are not aiming to teach every possible idea in the language but only a "Beginners" menu.

Over the past six months, work has been ongoing in filling that gap for LaTeX: `learnlatex.org`. The aim of this new site is to provide a carefully-curated set of resources for beginner LaTeX authors, with integrated use of an online LaTeX environment and demonstrations accessed directly from these lessons. The scope of these learning resources is focussed, and with the aim of offering the material in bite-sized chunks.

2 Existing resources

There are already many online resources for learning LaTeX, as a simple web search will reveal. The top hit at present is Overleaf's "Learn LaTeX in 30 Minutes" (`https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes`), which covers many of the core ideas with a series of examples. The second is `https://www.latex-tutorial.com`, which takes the step of dividing up the lessons into a series of pages. There are then more "traditional" resources, including the long-standing LaTeX Wikibook (`https://en.wikibooks.org/wiki/LaTeX`), and of course many excellent printed LaTeX guides.

The Overleaf page is notable as Overleaf provides a full LaTeX system online, and is used by many people as their LaTeX editor/system. On their teaching page, the examples are given as code blocks but to run then, a separate tab needs to be opened. That has the positive of looking exactly the same as any other document edited on Overleaf, but means leaving the teaching page. The other leading hits for learning LaTeX require that the learner copies or types out the examples by hand.

In contrast, users learning other programming languages can today often start on websites that let them try out the system *in the page*. That means no copy-pasting, no need to move to other tabs, and no need to download and install software. As mentioned above, `LearnPython.org` is an excellent example, forming part of a family of around half a dozen sites using the same overall framework to teach a set of modern languages.

The `learnlatex.org` project was started with the aim of combining existing best-practice teaching on LaTeX with a strong focus on interactive, online, training. That required tackling three things: the content, the website structure, and the online element.

3 Writing content

The starting point for writing the content was long-standing material developed by Nicola Talbot and used by the UK TeX User Group (UK-TUG) for face-to-face lessons (`https://github.com/uktug/latex-beginners-course`). After assembling a small group of volunteers, we began by looking at this curriculum, the content of other sites and discussions we have all had with new(er) users. That led to a first set of section titles, which we adjusted as new ideas arose.

We then started drafting the content, taking existing notes plus new material to deliver around 15 lessons.[1] As we worked through these, we found we had more to say than was reasonable to cover in a focussed set of lessons. We also felt there was good content that did not belong in the basic lessons, but did belong somewhere. That led to a split of each lesson into two parts: the basics and "going further". (We'll look at the file structures below, and how they build the site.)

As well as the content and demonstrations, we've worked hard to make sure that the lessons are up-to-date and accurate. That's led to discussions with the LaTeX team about for example `\label`, which is reflected in the lesson content and means that we are confident the site covers *best practice*.

4 Website structure

Based on experience with `texfaq.org`, we knew that GitHub Pages, which lets you write webpages in Markdown, was a good place to look to host the site.

[1] We've grown to 16 lessons, as there was a strong argument to add one on errors.

This means that multiple authors can easily work with the sources without needing to set up a complex build system: the pages can be edited on the GitHub site, so a local Git installation is not even required.

We started with a simple structure, with an index and 15 `lesson-XX` files (all in Markdown, of course). GitHub Pages runs a system called Jekyll, which turns those files into the full site. That's done using various bits of template plus some scripting, which can all be user controlled. As the site has grown, we split the extra content into `more-XX` files, which can then be linked automatically to their "parent".

Not long after we started work, the question of translations came up. We'd started with the idea of just working in English, but we didn't want to box ourselves in. So we moved all of the content into a directory called `/en`, and started exploring how to add a language selector. To help with that, we made some stub files that were marked as being in a few languages, then got at least the page titles translated by real people. It turns out to be reasonably easy to add a selector, and there's already been interest in translation, so we have the site in English, Vietnamese and Portuguese already, with Spanish, German and Japanese in the pipeline.

Linked to translations, we also realised that while most of the lessons stay the same for every language, we needed a place where language-specifics could be covered. That came up when we were asked about Japanese; they need vertical typesetting, which of course is very different from what we need for English. So a place for non-translated pages has been added: the content and number of pages there is down to the translators.

5 Online LaTeX compilers

As outlined above, a key aim in developing the site has been to allow examples to be run as close as possible to "in the page": that has led us to explore a range of options in this area. We wanted to have a full LaTeX system available, which mean that systems such as MathJax (`mathjax.org`) or even MiniLaTeX (`minilatex.app`) were not suitable.

5.1 Processing examples "in the page"

It is possible to run TeX in the JavaScript engine in the browser and initially we did some experiments with `texlive.js` but such systems are hard to keep up to date and currently provide only pdfLaTeX, not other engines such as LuaLaTeX. So the decision was made to use a server which ran LaTeX, returning the generated PDF to the web browser. An important consideration when choosing the technology here was that the examples should cover a range of languages and so a range of TeX engines should be available, notably pLaTeX, LuaLaTeX and XeLaTeX in addition to pdfLaTeX.

During initial development of the site, we mostly used the servers `latexonline.cc` and latex-on-http (`https://latex.ytotech.com/`). However, in the end we developed a server `latexcgi.xyz` specifically tailored to the requirements of the site.

The most noticeable distinguishing feature of the LaTeX CGI server is that it does not directly return the generated PDF but instead returns a call to a locally hosted copy of the `PDF.js` (`https://mozilla.github.io/pdf.js`) JavaScript library. This ensures consistent behaviour, notably on mobile browsers which typically do not include a built-in PDF renderer. The server does support options to return the PDF directly, or to return the log file, even of successful runs.

LaTeX CGI supports a range of LaTeX formats: `lualatex`, `pdflatex`, `xelatex`, `uplatex`, `platex` and their `-dev` variants. It also supports `biber`, `bibtex`, `pbibtex` and `bibtex8` and `makeindex`.

The server has installed a full copy of TeX Live 2020 and runs in the default "restricted shell escape" mode. This means that packages such as `imakeidx` that make use of the allowed shell commands are available.

Currently `latexcgi.xyz` is hosted as a virtual machine on Amazon's EC2 service, which is available in the *Free Tier* for one year. Long term hosting has yet to be finalized.

Using a dedicated server gives some flexibility in the technologies used. One possibility would be to use Docker images such as the TeX Live images provided by Island of TeX (`https://gitlab.com/islandoftex/images`). This would likely be necessary if we decide to offer more programming features, in particular allow TeX to be run with shell escape enabled, although currently the intention is to match the default (restricted shell escape) that matches a typical user's system.

5.2 Online TeX processing systems

All the systems mentioned in the previous section have the feature that they require no login or pre-registration, which is convenient but means that any edits to the documents are lost when the user moves off the page. The site therefore offers a second alternative to access a full online TeX system provider, currently Overleaf.

If the **Open in Overleaf** button is used, a new project is opened in Overleaf in a new tab in the

browser. The user can edit the example in Overleaf, and save the project if desired, to return to it later.

The Overleaf API was extended for this site, to allow multiple-file projects to be uploaded in this way.

Other TEX editing systems could be used but a requirement is a public API that starts a project via an HTTP request rather than using the web page menu interface.

6 What's next

The web design we are using is very basic: just what comes out-of-the-box with GitHub Pages. To help users navigate, we need proper development of a full site. That's beyond the expertise of the site team, but we have been able to raise funds to employ a professional web developer. He's looking at design both in terms of appearance and structure: things like mobile accessibility, metadata, etc., are all really important.

There's also still more to do on the content, in particular working out if we need more formalised exercises or lesson summaries. That likely needs user feedback, for which we need *your* help. The site will work best if it's promoted to potential users, and readers of *TUGboat* are we hope well-placed to recommend it. Of course, to do that, we'd encourage you to read the site, give feedback and perhaps even suggest some improvements!

◇ David Carlisle
Oxford, United Kingdom
d dot p dot carlisle (at)
gmail.com

◇ Paulo Roberto Massa Cereda
São Paulo, Brazil
paulocereda (at) gmail.com

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

A review of learnlatex.org

Jennifer Claudio

LearnLATEX (learnlatex.org) is an instructional guide to help new users of LATEX become familiar with creating their first documents and to introduce them to robust features that could be lacking or inefficient in popular word processor software. This review will address the following specific questions about the site:

- Is it pedagogically effective?
- Does it provide what a LATEX newbie needs to know?
- Does it encourage "best practices" and avoid harmful ones?
- Is it clear about what kinds of materials are supported?

An example screenshot from the site is shown at the end of this review, with the caveat that the site's graphic design has not yet been developed.

1 Is it pedagogically effective?

At first click, the LearnLATEX site is clean and welcoming, a good sign for new users. In Lesson 1, the site gives an overview of what LATEX is, how it works, and why it differs from WYSIWYG interfaces. Each of the modules includes options to learn more about the topic or go to the next lesson. The user is invited to follow along with examples by seeing the on-site display, opening in an editor, or by using a web resource such as Overleaf.

To address the question of pedagogical efficacy, though, this site assumes a sufficient inherent motivation from the user to complete each of the exploratory tasks. The methods of presenting information are indeed logically sequential and streamlined, but a user must want to complete tasks if his or her intention is to learn the content. The placement of each illustrative example and links to Overleaf (overleaf.com) and LATEX Online (latexcgi.xyz) are convenient and facilitate active engagement by the user, which is also pedagogically functional.

Each lesson has at least one exercise for users to try. Once again, a motivated user would do these for the sake of learning. To encourage the fraction of users exploring the LearnLATEX site who are less motivated, perhaps a checklist of "Things to try" would draw more strongly upon the strings of human desire for task completion. Another suggestion may be to include things to try that deliberately lead to errors, followed by a quick set of things to notice and consequently correct.

2 Does it provide what a LaTeX newbie needs to know?

For a user who needs to produce a standard document, LearnLaTeX provides enough starting points to populate text and try variations. It also provides relevant background and formatting tips that might be specific to a publication style, such as forced vertical or horizontal spacing or paragraph skipping rather than indentation. The site does focus on producing the article class, so a new user who wants to produce slides would need to either find support for that class or make creative decisions within the article class.

Inclusion of the math mode is an important section of this site, and reference to additional resources such as Detexify (`detexify.kirelabs.org`) are certainly helpful. The robustness of mathematical typesetting is seemingly a primary incentive for a person to use LaTeX rather than a word processor.

A great bottleneck for new users to LaTeX is neither an issue about raw difficulty nor inability to use markup functions. The perceived time sink (which is sometimes an actual time sink) needed to troubleshoot problems can be undesirable to a user who might be able to more quickly perform a task using software with which they are familiar. The result is that the potential user does not become any more familiar with LaTeX, and does not use it for future tasks unless required.

LearnLaTeX conveniently has a section titled *Dealing with Errors*, in which common errors for the typical user are examined. The supporting exercises to fix the errors illustrated provide an excellent method for recognizing and solving problems.

3 Does it encourage "best practices" and avoid harmful ones?

LearnLaTeX encourages "best practices" and tips for successful document creation. It provides insight into functional tips to help new users avoid pitfalls that can result in the aforementioned timesinks; for instance, mentioning that "It's important to finish a paragraph before changing the font size back." Almost all sections have such a guiding tip to avoid unnecessary troubleshooting down the line.

4 Is it clear about what kinds of materials are supported?

Regarding supported materials, it seems that LearnLaTeX walks its users through a variety of examples for file types, packages, databases, etc. The relevance of the supported materials naturally depends on the user's intended end product, but each section in LearnLaTeX provides clear details for how those are used.

5 Overall summary

The construction of this document (review) itself is perhaps testament that the LearnLaTeX site is useful to a new user who simply needs to create and submit a text-only document. A search function within the site could be helpful, though, since there were sections I wanted to be able to jump to quickly. This would eliminate the necessity to "flip through" pages or run web searches for help. Working in tandem with the Overleaf environment provides a convenient way to generate the markup and make tweaks as needed.

The LearnLaTeX site is usable as a self-paced course for a motivated LaTeX learner and seems sufficiently effective in its ability to guide a user from introduction to production. It does not yet, however, provide incentive for a non-user to explore it diligently unless the user has a specific need for LaTeX typesetting, which is a fair shortcoming of most instructional tools across all fields.

The site is currently or soon will be available in Chinese, English, French, German, Marathi, Portuguese, Japanese, and Vietnamese. (Some translations are still in progress.)

At the time of this submission, the website is still being edited and new features implemented. It is my personal hope, for the efforts that are being put into making LaTeX more accessible, that it will be adopted by a broader audience other than simply "those who must use it".

◇ Jennifer Claudio
San Jose, California
`claudioj (at) esuhsd dot org`

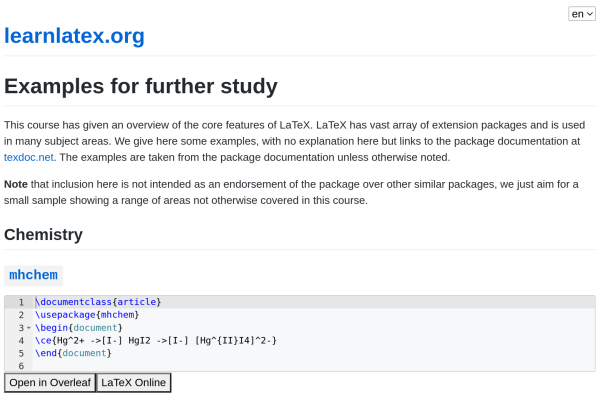


Figure 1: Interface for runnable example.

Quo vadis LaTeX(3) Team — A look back and at the upcoming years

Frank Mittelbach and the LaTeX Project Team

Abstract

This is a brief write-up of a talk given by the author at the TUG'20 online conference.

The talk touches briefly on the questions "where we are coming from" (we being the LaTeX Project Team), "where we are now" and then focusses on the LaTeX Project's plans for the upcoming years, which will primarily be focussed on providing an out-of-the-box solution for generating tagged PDF with LaTeX and will include gentle refactoring of parts of the core LaTeX and providing important functionality, such as extended standard support for color, hyperlinks etc., as part of the kernel.

This is a multi-year journey that we have just started and we will briefly explain the places this will take us through. At its end we expect that LaTeX users are able to produce tagged and "accessible" PDF without the need to post-process the result of their LaTeX run.

Contents

1 A quick walk through 30 years of history

In this section we take a short tour from the origins of LaTeX to the present day in order to better understand where we came from and its influence on how we see the future shaping.

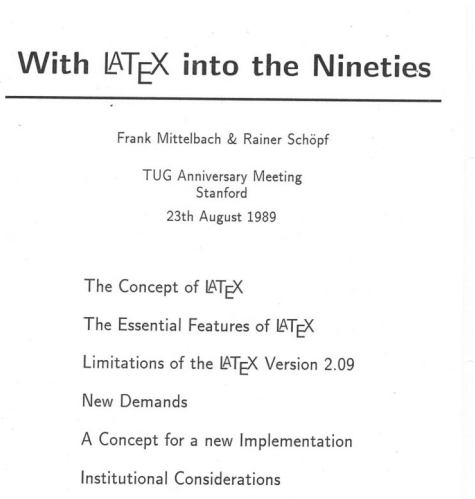


Figure 1: The title slide from 1989

The birth of the LaTeX project

A bit more than thirty years ago I gave my first international talk at the 1989 TUG conference at Stanford (Figure 1). There I lectured in front of Leslie Lamport and Don Knuth, boldly pointing out deficiencies of LaTeX 2.09 and what is needed to improve on it.

Our criticism wasn't new to Leslie, as we had sent him many bug reports and suggestions during the previous years. And after a long meeting following the talk, Leslie passed maintenance and future development of LaTeX on to Chris Rowley, Rainer Schöpf and myself. For more details on the events back then see the conference proceedings [9].

Leslie continued to work with us, discussing concepts and interfaces, but did not participate in any of the coding for a new version. By the time LaTeX 2_ε got released he had fully retired from working on LaTeX (except for sending in the occasional bug report like any other user).

Thus, this day in late August 1989 marked the origin of the LaTeX Project, later often referred to as the LaTeX3 project.

We were young (isn't that always the problem?) and had big plans, but it would certainly be impossible to turn even some of them into reality had we not had the fortune to soon recruit a number of additional members — influential in shaping LaTeX 2_ε and beyond.

Few of them will need introductions to anybody who has worked a while with LaTeX, but for the record here are the people beside Chris, Rainer and myself to praise or blame for LaTeX 2_ε and many of the packages that you are still using today: David Carlisle, Johannes Braams, Alan Jeffrey, Denys Duchier, Michael

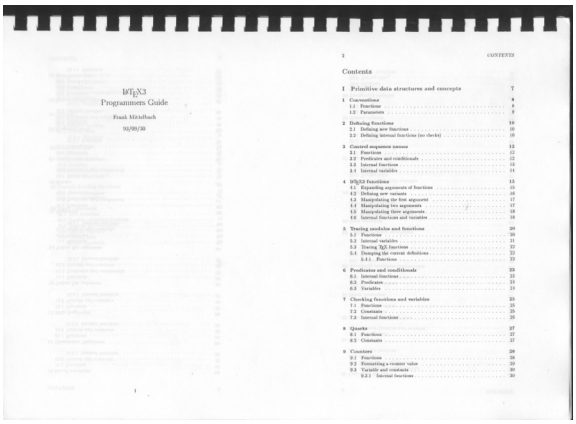


Figure 2: The 1993 LaTeX3 Programmer's Guide

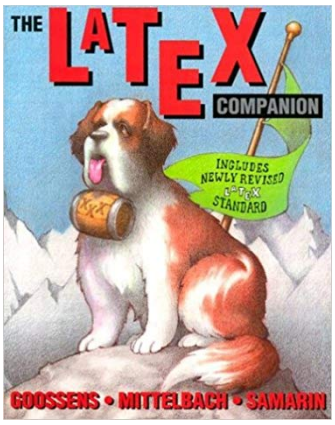


Figure 3: The "doggie" book: the first edition of the LaTeX Companion.

Downes and Robin Fairbairns. All got their hands dirty in the development of today's LaTeX and/or had a lasting influence on what later became expl3.

The first years

The early nineties were fairly productive years for the team and by around 1992 we had built a complete kernel for a new LaTeX3 system. It was able to compile its own user manual. Figure 2 shows a version from 1993.

In my talk I titled the slide showing this picture "A fully working (useless) LaTeX3 kernel". The reason is that we found out to our dismay that there was serious danger of dying of excessive caffeine consumption while waiting for even a small document to compile (let alone something like that guide).

Basically we were just several computer generations too early (or lousy programmers, or both). We invented, for example, a system comparable to Cascading Style Sheets (CSS) — long before that appeared in browsers. But all these ideas required much more computing power than was available on typical machines back then.

So in the end we gave up and decided that it was a nice but impossible dream.

A new LaTeX version

So instead of continuing with LaTeX3 we cleaned up all the extensions and improvements we had made for LaTeX 2.09, developed a graphics and color abstraction and bundled everything under the name LaTeX 2_ε. This was then promoted as the "newly revised LaTeX standard".

Leslie wrote an updated LaTeX manual [3] and Michel Goossens and myself with the help of Alexander Samarin produced the first LaTeX Companion [2], also known as the Doggie book to many LaTeX users (Figure 3).

LaTeX 2_ε had a fairly shaky start, largely due to the fact that a small but vocal minority loudly argued against using it and suggested to stay with LaTeX 2.09 instead. The main reason given was that "nobody needs these new 8-bit fonts with precomposed foreign characters and that they take a huge amount of unnecessary space on your hard disk".

However, in the end it took off like a rocket, largely because of all the other goodies it offered — solving many of the problems people were struggling with in the past. If you look through the Web for LaTeX 2_ε you will find a large number of books in various languages that appeared in the following years, clearly showing that there was a high level of interest in the software.

For us it was an important lesson to learn how close success or failure can be, if you have a small but vocal group opposing you.

Highlights of the following decades

Presenting any reasonably complete account of the works of the LaTeX team throughout the years would fill many pages, so here we restrict ourselves to only a few highlights that are relevant for what we intend to do in the future.

Around 1997

At some point we decided to release some of the work we did for LaTeX3 as a package on CTAN named expl3, mainly to preserve it but also because computers had gotten faster, and it seemed that the code could become usable after all at some point in the future.

The new millennium

That action got younger people interested, and first Morten Høgholm and later Will Robertson and Joseph Wright pushed for a complete overhaul of the

Figure 4: The expl3 logo

language. This happened in several phases between 2005 and 2012; see [12, 4] for more details.

Large application packages, such as fontspec, siunitx and others, got written in expl3. At some point the language was evidently in a reasonably stable state and we announced it as fit for general use [13].[1]

Now with a stable expl3 around 2014 we started promoting it and one of the actions was to use a logo for it, which was designed for us by Paulo Cereda — a lovely hummingbird pecking at the "l" (Figure 4).

To indicate that we are moving into new waters I pushed for using the hummingbird also as the official new logo for LaTeX and at some point later we made the switch.

Initially there had been some concerns to make the LaTeX "brand" unrecognizable if it isn't associated with some kind of a lion, but in retrospect it seems fairly clear that the logo was positively received and there is no question these days that this particular bird represents today's LaTeX.

A change in policy

A big step was taken in 2015 when we announced a new bug-fix and enhancement policy. Until then the LaTeX 2_ε format was essentially kept unchanged. Even serious bugs were either not fixed at all, or fixed by adding the fix to the package fixltx2e that one could or could not load as desired.

This meant great stability but it also meant that only the few people who added fixltx2e would benefit from the fixes, while the great majority would stay with the buggy version. In the beginning this was fine but over time it became a burden because packages have to provide alternative code paths based on fixltx2e being loaded or not. We therefore switched to the approach that fixes get applied by default (i.e., everybody receives them) and instead now offer a way (though a rollback mechanism [5, 7]) to opt out, if necessary.

Thus what happened in 2015 was that the accumulated fixes previously in fixltx2e got moved into the LaTeX kernel and the package reduced to an empty shell, unless you used it with an old LaTeX format.

Around that time we also started to bring external developments into core LaTeX. For example, we officially added support for LuaTeX to the kernel and took over the maintenance and development of amsmath from the American Mathematical Society.

Managing future enhancements

But we also went a step beyond bug fixes and integrations. To prepare for future developments we wrote a new testing and distribution environment (l3build [8]) that has been used by us to maintain the kernel sources, and over time also by many other package developers around the world.

A relatively recent activity was to arrange with the major distributions, i.e., TeXLive, MacTeX and MiKTeX, to provide so-called "LaTeX development releases", allowing users and package developers to test pre-releases of LaTeX with ease [6].

We also announced that necessary enhancements to the code (to keep it relevant) would be presented from now on in most cases as opt-out rather than opt-in solutions.

A good example for this policy change is the switch from legacy 8-bit code pages to Unicode, or more precisely to the UTF-8 encoding. This happened in 2018. With the LaTeX release in that year the default input encoding for LaTeX became UTF-8, and in retrospect it is fair to say that few people have noticed any ill effects with their document and had to apply the opt-out. Most people only noticed — if they noticed the change at all — that they could finally use Unicode characters in their documents without problems, a feature that was badly lacking in LaTeX previously.

2 Activities in 2020

Two important changes happened in the spring 2020 release of LaTeX:

- One was a long overdue modernization of LaTeX's font selection scheme to better support all the new high-quality OpenType fonts;
- The other was described in the *LaTeX Newsletter* as "improved load-times for expl3".

Why is the second bullet of any importance? At the time of the release it was indeed nothing more than what it said: users with documents loading expl3 (to begin with, all XeTeX or LuaTeX users) experienced noticeably faster processing times.

[1] The name expl3 stands for "**EX**perimental **P**rogramming **L**anguage (LaTeX) **3**" but it was kept even after it had long ceased being experimental.

But its importance lies in the fact that it marks the end of one era and the beginning of a new one. LaTeX now greets you with

```
LaTeX2e <2020-02-02>
L3 programming layer <2020-06-18>
```

and this means that thirty years after first dreaming about it, LaTeX finally comes equipped with the LaTeX3 programming layer included as part of the format.

3 A look at the future

LaTeX has stayed surprisingly relevant given that its original design dates back to the 1980s.[2] It has, however, limitations, some due to the underlying engine and some due to design decisions made in the past.

Important areas for urgent improvement

Perhaps the most important limitation is that until now LaTeX concerned itself only with producing a "printed result" with paper as the ultimate output medium in mind. Any other usage is either not supported or not directly supported. However, for quite a while now, other usage has become increasingly important. Many documents are never printed or printed only as a secondary action.

LaTeX 2$_\varepsilon$ added some support for graphics and limited color printing, but otherwise followed the same paradigm. Hyperlinks and other Web publishing support are layered on top, not as integral parts of the design.

As a notable example, the hyperref package has to redefine a larger number of LaTeX's internals and many commands of other packages to be able to achieve its goals and even so is often enough only able to do so by imposing restrictions. Other packages need to patch the same areas, resulting in conflicts and limitations.

Another important issue is that LaTeX very carefully throws away the wealth of structural information it has at its disposal while producing output pages. As a result a PDF or DVI file produced by LaTeX is just a stream of positioned glyphs without much structural information preserved.

If your intention is only to print that document, then this is all that is required, but if you want to produce, say, an accessible PDF document, then a significant amount of structural information and other data has to be embedded into the final output document to guide screen readers, etc., or adhere to the PDF/UA (Accessibility) standard. At the moment this requires extensive manual labor and processes that often have to be repeated after making even minimal changes to the LaTeX source.

Project(s) for the upcoming years

With the challenges outlined in the previous section in front of us we are focussing on a number of areas to address them:

- Embrace and integrate more functionality from existing packages into the LaTeX kernel;
- Provide extended and unified color management, with graphics and font(glyphs) integration;
- Provide standard interfaces for functionality currently available only in an ad hoc way, or not available at all;
- *Enable LaTeX to automatically produce tagged PDF.*

We plan to integrate important functionality from existing packages directly into LaTeX so that it is directly available for user and package writers through standard interfaces. Examples for this are hyperlinks and colors, as already mentioned, but there are several other areas we are looking at.

In addition, we plan to provide standard interfaces for some important capabilities that are currently not available at all or only in rudimentary and ad hoc fashion. An example for this is the hook management system that is planned for the next LaTeX release in fall this year.

Finally, the list contains a one-liner about producing "tagged PDF", which hides a huge project — we will discuss this below.

A focus change — modernize LaTeX through gentle refactoring

When we set out in 1989 to improve LaTeX 2.09 and produce a new version (a.k.a. LaTeX3) the LaTeX universe was largely defined by the software provided by Leslie Lamport and a rather small and manageable number of packages by others. The reason being that it was not at all easy to build applications on top of LaTeX 2.09 and, of course, LaTeX was only a few years in use back then.

When LaTeX 2$_\varepsilon$ was released in 1994 it solved many of the problems we had initially criticized, even though it wasn't the system we had envisioned — one with a clear separation of user, designer and programmer levels and facilities, which we simply couldn't make work with the existing computing power of those days.

However, LaTeX 2$_\varepsilon$ offered a package management system with \usepackage, command declaration with optional arguments and other goodies for

[2] Or the early 1990s if you think of LaTeX 2$_\varepsilon$ as the starting point for today's LaTeX.

users and package developers and so over time people started to provide more and more packages for LaTeX that filled the needs of any niche and nowadays several thousand packages for LaTeX are on CTAN.

The increase in the breadth of the software usage over the years made it more and more unlikely that producing a standalone LaTeX3 next to an existing LaTeX 2ε would gain any traction. It would naturally start out with a very limited scope (because many existing packages would not work with it) and would therefore be unsuitable for most serious usage. But that in turn would mean that as kernel developers we would not get the necessary feedback that ensures that the provided features are meeting the needs of the users and as a package developer there would be no incentive to provide new packages for a new system that isn't widely used — the usual chicken and egg problem.

We have therefore decided that there will be no separate LaTeX3 product in parallel to an existing LaTeX 2ε. Instead, we will approach the modernization through some gentle refactoring of LaTeX to reach the same target, but in smaller steps.

If you look back at the history outlined earlier, you will see that this journey has already started in 2015 with the new bug-fix policy and the rollback mechanism, which was then followed by the switch to UTF-8 to keep LaTeX relevant.

The strategy we are following here can be outlined in a number of main bullet points:

- Use the L3 programming language to implement all new kernel code now that it is available;
- Replace existing kernel code (over time);
- Keep focus on reliability and compatibility;
- Collaborate with package writers/maintainers to ensure compatibility with kernel changes.

An example of our new strategy is the implementation of a hook management system for LaTeX, which will be introduced in LaTeX in the 2020 fall release.

Hook management as an example

In the past LaTeX offered just a few heavily used hooks, for example, `\AtBeginDocument`. Every other alteration or addition made by a package was done by overwriting existing kernel code, leading to all kinds of known issues.

With the new hook management system, the LaTeX kernel and many packages will get a larger number of hooks in which other packages can add code in a controlled manner, avoiding the need for patching commands. The new system provides standard interfaces for declaring and using hooks, including ways to order code added to hooks by different package in order to resolve package loading problems, and plenty more.

The new system is written in the L3 programming language (the source file is `lthooks.dtx`), but the interfaces are offered in a way that they can be used in all packages, i.e., they do not require the package to be written in expl3 and thus can be retrofitted into updates of legacy packages easily.

The individual hooks provided by the kernel in the first release replace ad hoc solutions in specific areas as provided by packages such as atbegshi, everyshi, atveryend, etoolbox, filehook and others. In future releases, more parts of LaTeX will see hooks added.

Thanks to the *LaTeX development format* concept mentioned above, the new hook management code is already available for testing to anybody interested — which we strongly encourage. As any change to LaTeX will inevitably have ripple effects which need sorting out, such pre-testing is an important part of the overall strategy, to resolve as many problems and borderline cases as possible before new code shows up in the main release.

For the same reason the LaTeX team is actively checking across the huge set of packages supplied in TeX distributions for possible conflicts and working with other developers and maintainers if updates are necessary due to upcoming LaTeX kernel changes. In this particular instance, it was necessary for a handful of packages that patched into existing internal LaTeX commands in places that have been unavoidably changed to support the new hooks.

4 The tagged PDF project

This project is the LaTeX team's answer to the need for preparing LaTeX to uses other than printing on paper. The main goals of this project can be summarized as follows:

- Provide functionality to automatically produce structured PDF, without the need for user intervention or post-processing;
- Provide the necessary interfaces for producing PDF enhanced by features such as "alternative text" (to comply with standards such as PDF/UA).

While the project focusses on PDF as the primary output format, the functionality that needs to be developed will be equally applicable when targeting other output formats that require structured data to be present, e.g., HTML, XML, and new formats such as HINT currently being developed [16, 17].

Background and project status

There has been groundbreaking work done by Ross Moore and others [10, 14, 11] in the last years in the quest for enabling LaTeX to produce "accessible" or more generally "structured and enhanced" PDF.

The unfortunate problem which all these attempts have run into is that it is next to impossible to patch current LaTeX and all needed packages and still obtain reliable and stable results.

A system based on patches is by its nature very fragile, because any change in the patched code will break the system — which will happen regularly if significant patching is needed, as is the case here. In addition, all solutions to date need to enforce severe restrictions on the document content and even then require the user to do serious manual work — largely because of missing machinery and interfaces in LaTeX.

Our plans are therefore to continue learning from this prior work and provide the necessary interfaces directly in LaTeX, so that fragile and incompatible patching is no longer necessary. Some of our initial work in this regard is documented in [15, 1].

What we have undertaken so far with respect to the "Tagged PDF project" is to produce a feasibility study and develop a detailed project plan for reaching the project goals. This is a multi-year undertaking split into six phases and how long it will take will depend in part on the financial backing for the project, i.e., it depends on how much of the work has to be done in our spare time and how much of the development work is financed by sponsors, so that we can have some people work full time on the necessary work.

We are therefore pleased to be able to say that Adobe is sponsoring a fair portion of the estimated project costs, though we hope to attract further industry sponsors and organizations interested in the subject, in order to keep the timeline at a reasonable length.

Project phases and timeline

The project is tentatively divided into six phases progressing in parallel to the LaTeX release cycle; that is, each phase is expected to require one or more LaTeX releases, depending on how much time we can devote to the necessary work.

The deliverables of each phase are expected to be directly applicable to LaTeX users (and developers) so that we can get immediate feedback but also make tangible progress.

Overall, depending on the available financial support, the project timeline is expected to take between three and five years.

Phase I — Prepare the ground

This phase is already well under way and one important deliverable is the introduction of a general hook management system, discussed earlier.

Phase II — Provide tagging of simple documents

The main goal of phase II is to provide automatic tagging of simple documents, excluding more complicated structures such as mathematics and tables. In this phase workarounds are needed for code that will be adjusted later.

This is delivered as a prototype implementation in form of an add-on package.

Phase III — Remove the workarounds needed for tagging

The main goal of phase III is to extend the coverage of automatic tagging and to remove workarounds that were initially necessary to provide a working prototype.

Phase IV — Make basic tagging and hyperlinking available

The main goal of phase IV is to incorporate all the code currently in the prototype packages into the kernel itself. This needs to be done very carefully and cautiously as there should be no negative impact for users processing legacy documents. This is why we expect to need at least a full release cycle for this.

Phase V — Extend the tagging capabilities

With basic tagging available the focus of phase V lies in providing extended support for tagging by adding tables and formulas to the supported elements.

Furthermore, interfaces for specifying alternate text will be developed and added to all relevant elements.

Phase VI — Handle standards

Finally, phase VI will focus on providing additional support for the relevant PDF standards (as far as this is possible using LaTeX directly, without post-processing the resulting PDF), and adding kernel support for outlines and associated files.

Parallel work

In addition to the six phases (which contain tasks that are largely understood from a technical perspective) there are a number of tasks that require research. These will be carried out in parallel to the other work.

Depending on their outcome the structure of the later phases might need some alteration or extension.

5 Stay tuned

Clearly this article provides only a short glimpse of our plans for the immediate and mid-term future. The feasibility study for the tagged PDF project and its implications and dependencies, for example, is a forty page document and touched upon in this document in a few sentences. In the near future we intend to publish this study and more details both on the plans and on our intermediate results.

As a first result from Phase I, you can already now take a look at the new hook management system and provide your feedback for consideration before it get officially introduced in the fall release of LaTeX. With an up-to-date LaTeX installation the relevant commands are:

`texdoc lthooks-doc` (for documentation)
`pdflatex-dev` *yourfile* (for testing)

References

[1] U. Fischer. Creating accessible pdfs with LaTeX. *TUGboat* 41(1):26–28, 2020. https://tug.org/TUGboat/tb41-1/tb127fischer-accessible.pdf

[2] M. Goossens, F. Mittelbach, A. Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, MA, USA, 1994.

[3] L. Lamport. *LaTeX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1994.

[4] LaTeX Project Team. LaTeX3 news, 2009–. https://latex-project.org/news/latex3-news/.

[5] LaTeX Project Team. The latexrelease package, 2018. https://ctan.org/pkg/latexrelease.

[6] LaTeX Project Team. LaTeX news, issue 30, October 2019. *TUGboat* 40(3):251–254, 2019. https://tug.org/TUGboat/tb40-3/tb126ltnews30.pdf

[7] F. Mittelbach. A rollback concept for packages and classes. *TUGboat* 39(2):107–112, 2018. https://tug.org/TUGboat/tb39-2/tb122mitt-rollback.pdf

[8] F. Mittelbach, W. Robertson, LaTeX3 team. l3build — A modern Lua test suite for TeX programming. *TUGboat* 35(3):287–293, 2014. https://tug.org/TUGboat/tb35-3/tb111mitt-l3build.pdf

[9] F. Mittelbach, R. Schöpf. With LaTeX into the nineties. *TUGboat* 10(4):681–690, Dec. 1989. https://tug.org/TUGboat/tb10-4/tb26mitt.pdf

[10] R. Moore. Ongoing efforts to generate "tagged PDF" using pdfTeX. *TUGboat* 30(2):170–175, 2009. https://tug.org/TUGboat/tb30-2/tb95moore.pdf

[11] R. Moore. Implementing PDF standards for mathematical publishing. *TUGboat* 39(2):131–135, 2018. https://tug.org/TUGboat/tb39-2/tb122moore-pdf.pdf

[12] LaTeX. Project Team. LaTeX news, issue 17. *TUGboat* 28(1):24–25, 2007. https://tug.org/TUGboat/tb28-1/tb88ltnews.pdf

[13] LaTeX. Project Team. LaTeX3 news, issue 9. *TUGboat* 35(1):22–26, 2014. https://tug.org/TUGboat/tb35-1/tb109l3news.pdf

[14] C. V. Radhakrishnan, Hàn Thế Thành, et al. Generating PDF/X- and PDF/A-compliant PDFs with pdfTeX — pdfx.sty. *TUGboat* 36(2):136–142, 2015. https://tug.org/TUGboat/tb36-2/tb113radhakrishnan.pdf

[15] C. Rowley, U. Fischer, F. Mittelbach. Accessibility in the LaTeX kernel — experiments in Tagged PDF. *TUGboat* 40(2):157–158, 2019. https://tug.org/TUGboat/tb40-2/tb125rowley-tagpdf.pdf

[16] M. Ruckert. The design of the HINT file format. *TUGboat* 40(2):143–146, 2019. https://tug.org/TUGboat/tb40-2/tb125ruckert-hint.pdf

[17] M. Ruckert. The HINT project: Status and open questions. *TUGboat* 41(2):208–211, 2020. https://tug.org/TUGboat/tb41-2/tb128ruckert-hint.pdf

◇ Frank Mittelbach and the LaTeX Project Team
Mainz, Germany
frank.mittelbach (at) latex-project dot org
https://www.latex-project.org

The HINT Project: Status and open questions

Martin Ruckert, Gudrun Socher

Abstract

The HINT file format is intended as a replacement of the DVI or PDF file format for on-screen reading of TEX output. We give an overview of the current state of the project and solicit answers to open questions that might influence further development.

1 Current state

1.1 Version 1.0

Version 1.0 of the HINT file format [4] was published in August 2019 and presented [3] at the TUG 2019 meeting. In March 2020, the first version of the HiTEX engine and two HINT viewers, one for Windows and one for Android, went online on `http://hint.userweb.mwn.de`. The performance of these programs was better than expected in time as well as in space.

HiTEX runs as fast as other implementations of TEX; after all, it skips the breaking of paragraphs into lines and the building of pages. The size of the HINT files it produces are similar to the size of PDF files produced from the same source. Even smaller files can be expected in the future because the current implementation of HiTEX does not use the more compact "text format" defined as part of the HINT specification.

The first viewer for version 1.0 HINT files became operational in fall 2019. It runs under the Windows operating system using the WIN32 API. While it is not optimized for speed, its performance is still good and the rendering time does not depend on the file size nor on the position of the page inside a large file. Large pages on big screens at tiny font sizes are shown with a small, noticeable delay, but average pages — comparable to lettersize paper at 10pt — pop up at once.

When we started to build the first viewer for the Android operating system, we were aware of the limited computational resources available on mobile devices and curious how the user experience would turn out. We decided to use Open EGL to delegate the rendering of glyphs to the GPU and were positively surprised that the new implementation — even on low-cost mobile phones — clearly outperformed the Windows implementation running on a standard PC. It turned out that the computational demands of the TEX-based backend are very low and fast rendering depends almost exclusively on the ability to accelerate the transfer of bitmaps to the screen. As a side effect, a group of (lazy) students, charged with implementing various features of the user interface as part of a computer graphics course, successfully used the full power of the TEX backend to implement a two-finger zoom gesture: re-rendering the complete page with every movement of the fingers including line breaking and page building. Just to demonstrate the backend performance this unusual way of zooming is left as an option — named "TEX Zoom" — in the Android implementation.

Both viewers, for Windows and for Android, rest on a shared backend, written as a literate program in CWEB (3475 lines of change files for TEX plus 6109 lines of CWEB code).

The Windows viewer is written in C (1056 lines plus 176 lines for print support); its user interface is mouse- and keyboard-based. The Android viewer is written in C++ (947 lines) with some embedded Open EGL and Java (1201 lines); its user interface is touch-screen based. In both cases, additional libraries are used for decompression and rendering of fonts and images.

1.2 Version 1.1

Since the publication in March 2020, new features have been added to the software and we expect Version 1.1 to be ready for publication in fall 2020. Unfortunately the Version 1.1 HINT file format is not compatible with the 1.0 Version. But the HINT project is a research project and it seemed reasonable to make these changes.

First, a new tag was introduced to indicate the current language. Furthermore, font descriptions now indicate the encoding. This information was not available in Version 1.0 because it is not needed to render HINT files on screen. Knowing the language and encoding, however, is very important for language translation or for text to speech conversion to make texts accessible to the visually-impaired.

Second, HiTEX was extended by special syntax to define page templates and the viewer backend was extended to use page templates. So now it is possible to display footnotes and floating insertions. In the process of implementing these features, the representation of insertions and page templates in the HINT file format was slightly redesigned.

The new Windows viewer now has a "Print" feature. Taking a TEX file that is designed for, let's say, letter paper, producing a HINT file from it, and printing it on letter paper should give exactly the same result as producing a PDF file and printing the PDF. The "Print" feature is not (yet) available on Android, because the interface between Android's print manager and Open EGL turned out to be challenging.

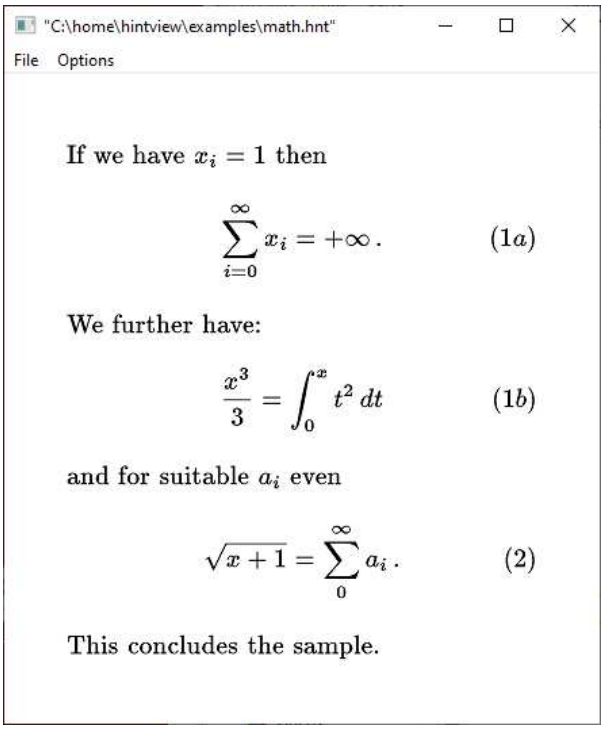
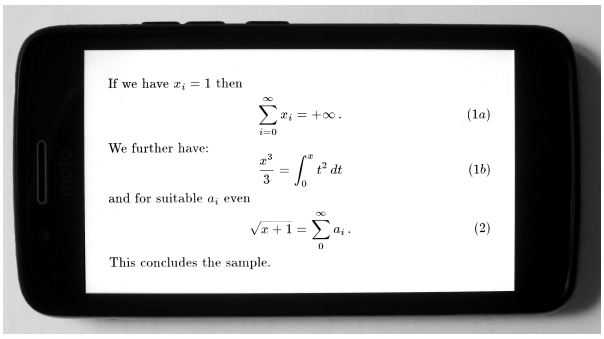


Figure 1(a): Some math displayed in the Windows viewer; screenshot at actual size on a 142dpi screen.



(b): The same document, in landscape mode on a mobile phone. The fonts in both images are the same size.

A few words about math support: because TEX's formatting of mathematical formulas does not depend on `\hsize` or `\vsize`, HiTEX does not require any special handling of math mode. The only exception is the positioning of a displayed formula and its equation number in a paragraph. Here HiTEX will encode the necessary information, such as the value of `\abovedisplayshortskip`, together with the formula and the equation number in the HINT file, and the viewer applies TEX's algorithm to position them correctly. Two examples are shown in figure 1.

The LATEX support of HiTEX is still incomplete because HiTEX is still based on Knuth's TEX distribution and the extensions of ε-TEX are not yet part of it. HiTEX's memory model is still based on Knuth's 16 bit pointers. We hope, however, that in the near future, HiTEX and the HINT viewers can become a standard part of the TEX Live distribution and as easy to use as any other TEX engine.

2 Open questions

Now to the questions. Two basic goals of the HINT project are:

1. The HINT format is independent of TEX. It should lend itself as an output format to all kinds of document processors.
2. HiTEX is source-compatible with other common versions of TEX like pdfTEX.

As a consequence, it is important to use common standards both inside and outside the TEX universe.

2.1 Language information and character encoding

Universally accepted standards have been defined for the world wide web to represent language information. The Internet Engineering Task Force (IETF) has defined in BCP 47 [1] tags for identifying languages: short strings like "en" for English or "de" for Deutsch, and longer ones like "sl-IT-nedis", for the specific variant of the Nadiza dialect of Slovenian that is spoken in Italy. A HINT file should contain these language tags to enable tools, for example a text to speech converter, to process a HINT file. The open question is: How can I obtain this information from a TEX source file?

The babel package is the de facto standard handling language selection in TEX. It provides a mechanism to map the tags from BCP 47 to TEX's language numbers and back again. Adding these tags to an output file is, however, less simple: When TEX's whatsit nodes with a language subtype arrive at the page builder, only the language number is left. I think all TEX engines that want to embed language information in their output would benefit from a simple standard mechanism; for example, adding a TEX string number pointing to the BCP 47 tag to the language whatsit nodes.

The situation with character encodings is similar. In a HINT file, text is represented as a sequence of numbers called character codes. HINT files use the UTF-8 character encoding scheme (CES) to map these numbers to their representations as byte sequences. For example, the character code "`0xE4`" (228 decimal) is encoded as the byte sequence "`0xC3 0xA4`". The same number `0xE4` now can represent different characters depending on the coded character set (CCS). For example in the common ISO-8859-1

(Latin 1) encoding the number `0xE4` is the umlaut "ä" whereas in the ISO-8859-7 (Latin/Greek) it is the Greek letter "δ" and in the EBCDIC encoding, once common on IBM mainframes, it is the upper case letter "U".

The character encoding is irrelevant for rendering a HINT file as long as the character codes in the HINT content section are consistent with the character codes used in the font file, but the character encoding is necessary for all programs that need to "understand" the content of the HINT file. For example, programs that want to translate a HINT document to a different language or for text-to-speech conversion.

The IETF has established a character set registry [2] that defines an enumeration of all registered coded character sets. The coded character set numbers are in the range 1–2999. This encoding number is required in a HINT file as part of a font definition. Is there a method to obtain this number in a standard way when processing a TEX source file? Again, all TEX engines that want to produce accessible output will need that information.

2.2 Images

This section is only about still images, not animated images. There are many different file formats to store images, but most of them exist only for historic reasons, marketing considerations, or patent rights. If you look around the web, a few image file formats are common: JPEG for photographs; PNG for clipart, button faces, and other simple graphical objects; and SVG for resolution-independent vector graphics. Image formats that are commonly used with TEX, like EPS or PDF, are not found among them. Further, TEX engines and TEX viewers support different sets of image formats. There are very good programs available to convert basically any image format into any other image format, and therefore it seems reasonable to define one basic set of image formats that are supported by any TEX engine and any TEX viewer. I assume that JPEG and PNG are most likely part of such a set. The open question is: What kind of image format should be used for vector graphics?

Among the different versions of SVG, probably compressed SVG Tiny is the only viable candidate for the HINT file format, since the HINT file format is specifically designed for mobile devices, such as eBook readers, with severely limited capabilities. But even for SVG Tiny, it seems there is no simple, high-performance library that would do the job. Of course there are interpreters for SVG, EPS, or PDF graphics, but unless you need such an interpreter anyway for viewing your TEX output, these interpreters add considerably to the footprint of the TEX viewer.

The only extra interpreter that is (planned to be) part of the HINT viewer is the FreeType library. It is needed for PostScript Type 1 fonts. Therefore the cheapest alternative would be a program that converts vector graphics to glyphs in such a font. Is this a viable idea?

Finally, someone might be able to answer this question easily, but I have not investigated it yet: What is the minimal set of primitives I have to implement in HiTEX to make the typical LATEX graphics packages happy?

2.3 Links

For an output format like HINT that supports on-demand page building, the usual method of books — and TEX — to implement references, namely page numbers, does not work. Therefore links are necessary to navigate large documents. Generating links is already common practice when TEX's output is a PDF file. But note, that for example the *TUGboat* sample article template starts with

```
\usepackage{ifpdf}
\ifpdf
\usepackage[...]{hyperref}
\else
\usepackage{url}
\fi
```

This is a clear indication that there is no common set of primitives to implement links across output formats. Is the only solution another `ifhint` package?

Many of the new primitives of PDFTEX write their arguments directly to the PDF output file. Is it not better to define a new, standardized subtype of whatsit nodes for representing links, together with a common set of primitives to generate these nodes, and postpone the generation of output format dependent code?

2.4 The viewer API

Unlike previous questions, this one is mainly, though not exclusively, HINT-specific. When viewing TEX output, be it a HINT file, a PDF file, or some other format used for example in a WYSIWYG editor, the viewer must support interactions between user and document. In simple cases this is just paging forward or backward, and in more sophisticated cases following a link or zooming. In contrast to DVI or PDF output, the document representation in a HINT file is more or less the representation that TEX uses internally for the document. Therefore the question of

how to interact with such a document might be of interest for all programs that interactively manipulate TEX documents.

The HINT project separates the document handling from the graphical user interface (GUI). It provides a generic backend program with a clear API that should be flexible enough to support any GUI frontend. Currently the following operations are supported: opening and closing a document, setting the size of the output area (true size and resolution), rendering the current page, obtaining the top-left position of the current page, and moving to a new page given the top-left or bottom-right position.

Other operations are under consideration. For example, consider that the user interface wants to implement searching for words. Opening a window and entering the search term is entirely the responsibility of the GUI. The backend needs to supply a function to find the position of the next occurrence of the search term. But just displaying a page that starts or ends with the given position is probably not what the user wants. A new function is needed that moves to a new page that contains the given position somewhere in the middle. Another function should probably be available to test whether the new position is already on the current page.

To interact with images, floating insertions, and links might require many more additional functions. Since there is no point in reinventing the wheel, we ask: Is there an established set of functions or patterns to accomplish such tasks which is as simple as possible and as powerful as necessary to support the user interfaces that — hopefully — will be written in the future?

2.5 Change files as literate programs

Large parts of the HINT project are written as literate programs using CWEB. The special situation with HINT is, however, that important parts of the code are taken directly from Knuth's TEX implementation, thus written in (the original Pascal) `WEB`, but with many, many small modifications. For example, TEX's dimensions become extended dimensions in HINT. An extended dimension is a linear function of `\hsize` and `\vsize` and hence every occurrence of `cur_val` needs to be supplemented by an occurrence of `cur_hfactor` and `cur_vfactor`. Similar supplements are needed for the table of equivalents and the save stack.

The traditional method for such modifications is to use change files or slightly less traditional, but more convenient, patch files. Large change files or patch files tend to be dull reading material up to the point where they must be considered unreadable. Two main problems affect the readability of change files: the lack of context and the necessity to order changes by their appearance in the original. The situation can be alleviated somewhat by using the `tie` program and organizing the changes into collections of related changes. For the HINT documentation, from these change files reasonable TEX output is generated. But overall, the result is still less than satisfactory, and this raises the question: Is there a good way to present changes to a literate program as a literate program?

3 Conclusion

Considering the scale of the project and the complexity of the involved software, the HINT project has moved forward with surprising speed. While there are still many open questions and missing pieces, the available HINT prototypes are already usable for small projects and provide a testbed to explore the advantages and the challenges of on-demand paging with TEX.

Version 1.1 will provide a more stable basis and offer enough flexibility to make HINT files a viable alternative to the DVI or PDF format for on-screen reading of TEX output. In the long run, however, a new document format will survive only if it is either widely used or if its infrastructure is sufficiently superior to existing formats. Neither is currently the case. Therefore the HINT project is still looking for partners in the industry that have the will and the necessary resources to turn the HINT project from a research effort into a product.

References

[1] E. A. Phillips, E. M. Davis. *Tags for Identifying Languages*. IETF Internet Engineering Task Force, Sept. 2009. RFC 5646, BCP 47. `tools.ietf.org/html/rfc5646`

[2] I. McDonald. *IANA Charset MIB*. IETF Internet Engineering Task Force, June 2004. RFC 3808. `tools.ietf.org/html/rfc3808`

[3] M. Ruckert. The design of the HINT file format. *TUGboat* 40(2):143–146, 2019. `tug.org/TUGboat/tb40-2/tb125ruckert-hint.pdf`

[4] M. Ruckert. *HINT: The File Format*. Independently published, August 2019. ISBN 978-1079481594. `hint.userweb.mwn.de/hint/format.html`

◇ Martin Ruckert
Gudrun Socher
Hochschule München
Lothstrasse 64
80336 München
Germany
`martin.ruckert (at) hm dot edu`,
`gudrun.socher (at) hm dot edu`

MiniLaTeX: A subset of LaTeX for the Web

James Carlson

Abstract

MiniLaTeX is a no-setup subset of LaTeX that can be rendered on the fly to HTML. One can use it to build web apps with true HTML display viewable on any device from smart phone to tablet to desktop. Typesetting occurs in real time, and error messages are displayed in-line in the rendered text. MiniLaTeX documents can be exported to standard LaTeX.

We describe (a) MiniLaTeX the language, (b) the main features of the document management application `minilatex.lamdera.app`, and (c) some of the technical work required to implement an on-the-fly LaTeX-to-HTML compiler.

A video version of this paper is at `https://youtu.be/TAIYpCc3VV0`.

1 Introduction

MiniLaTeX is a no-setup subset of LaTeX that comes with an on-the-fly compiler to HTML:

- **No setup.** Just begin typing. No preamble with `\usepackage`, `\begin{document}`, etc., is needed.
- **On-the-fly.** A typical editor for MiniLaTeX presents two windows: on the left is the source text, on the right is the rendered text. Changes to the source text are immediately reflected in the rendered text window.
- **Errors.** Errors are immediately flagged in color in place in the rendered text.

The compiler is written in Elm (`elm-lang.org`), a strictly typed language of pure functions. Elm is a good language for writing MiniLaTeX apps because (a) it is designed for building web applications and (b) it has an excellent, high-performance library of parser combinators (`package.elm-lang.org/packages/elm/parser/latest`), akin to Haskell's `parsec`.

Here are two links to apps that use MiniLaTeX.

- `demo.minilatex.app`: a simple no-signin app for experimenting with MiniLaTeX. Feel free to edit the text you find there, or clear it and write something new. Since there is no setup and since it is interactive, the demo app is also a good tool for learning LaTeX.
- `minilatex.lamdera.app/g/34`: This link takes you to some class notes, hosted on `minilatex.lamdera.app`.

`minilatex.lamdera.app`, while still in alpha test, is a full content-management system for creating, editing, and distributing MiniLaTeX documents. Some ways to access it:

- **Guest access.** Sign in to `minilatex.lamdera.app` as `guest` with password `minilatex` to explore documents that have been made public by their authors. Both the source and rendered text are available, so you can see how MiniLaTeX documents are written.
- **Registered user.** Create and edit documents on a desktop computer or tablet.
- **Smart phone.** Use in read-only mode on a smart phone. Students can read class notes and problem sets this way.

2 Features of MiniLaTeX

Below is a summary of MiniLaTeX features. See the manual (`minilatex.lamdera.app/g/21`) for more details.

- **Macros and Environments.** Environments include theorem, problem, definition, etc., as well as equation, align, verbatim, and more. The `\maketitle`, `\section`, `\cite`, `\eqref` macros and many others work as expected. Unimplemented macros are rendered verbatim, but colored red to indicate their status.
- **Macro definitions.** One can define macros for both math mode and text mode in MiniLaTeX.
- **Export.** MiniLaTeX documents can be exported to standard LaTeX, complete with the necessary `\usepackage`, `\begin{document}` ... `\end{document}`, and any needed macro definitions. Exported documents are ready to process with `pdflatex`. An example of an exported document compiled to PDF using TeXShop is at `noteimages.s3.amazonaws.com/anharmonic_oscillator.pdf`.
- **Images.** A macro
  `\image{URL}{Caption}{Format}`
  is provided to place images in a MiniLaTeX document. Provision is made to render images in exported documents. In addition, there is an `svg` environment for rendering SVG images from SVG source code.
- **Unicode.** MiniLaTeX accepts Unicode (UTF-8) input. More needs to be done to accommodate Unicode in exported documents.
- **Paragraph-centric.** MiniLaTeX is "paragraph-centric", meaning that the smallest unit of recompilation is the *logical paragraph*. A logical paragraph is either an ordinary paragraph or an outer begin-end block delimited above and below by a blank line.

- **Additions.** Besides the `\image` command, various other commands exist in MiniLaTeX but not standard LaTeX. These commands are provided with suitable macro definitions on export so that a MiniLaTeX document can always be compiled with standard LaTeX tools. Here are some examples (colors are grayscaled in the printed *TUGboat*; apologies). Text can be colored blue using the `\blue` macro: *I am feeling blue*. Text can be highlighted using the `\highlight` macro. The teacher said that all work on our class project is due by October 1. There is also a `\strike` macro: Please delete this ~~very bad word~~. For a complete list of additions, see `minilatex.lamdera.app/g/21`.

An important part of the MiniLaTeX project is to properly define the subset of LaTeX to be supported. The current rule of thumb for this is "good enough to write my lecture notes, class hand-outs, and problem sets". Feedback on this issue is much appreciated.

3 Some features of minilatex.lamdera.app

`minilatex.lamdera.app` hosts a content management system which supports creating, editing, and distributing documents from a searchable repository. Users can search by title, author, tags, etc. Documents can be collaboratively edited, shared by url, and versioned on Github through a simple user interface. Documents can also be exported to PDF.

In addition to the familiar `\href` macro, `\xlink` and `\ilink` are provided in MiniLaTeX. The `\xlink` macro is used to make a link from one document to another in `minilatex.lambdera.app`. Thus, one can say `\xlink{21}{Manual}` to make a link to document 21 with label *Manual*.

The `\ilink` macro is similar. It has the same syntax, but is used in constructing a page of links which functions as a table of contents or *index document*. In this way, one can assemble many documents into one to make a book. For an example, see the class notes at `minilatex.lamdera.app/g/34`. It is worth looking at the source to see how it is done.

4 The MiniLaTeX Compiler

The MiniLaTeX compiler consists of two parts, a *parser* and a *renderer*. The first is a function

$$\textit{parse}\text{: Source text} \to \text{AST},$$

where *AST* stands for *Abstract Syntax Tree*. This is a tree with nodes like

```
LXString "Pythagoras says"
```

and

```
InlineMath "a^2 + b^2 = c^2"
```

The tree thus expresses a grammatical analysis of the source text, identifying its "parts of speech" and putting them in relationship to one another. The second is a function

$$\textit{render}\text{: AST} \to \text{HTML}.$$

The compiler is the composite of these two functions:

$$\text{compile} = \text{render} \circ \text{parse}$$

The strategy that makes writing such a compiler feasible is *divide and conquer*. One writes the parser using parser combinators. One then constructs a function which renders the text-mode LaTeX to HTML, passing the math-mode text on to either MathJax (`https://mathjax.org`) or KaTeX (`https://katex.org`) for rendering.

Video: Making a LaTeX-to-HTML parser in Elm (`https://youtu.be/dmDA7iziSgs`).

4.1 MiniLaTeX's AST

Every value in a statically typed language like Haskell, ML, or Elm, has a *type*. Below is the type of the AST for the MiniLaTeX compiler. Writing down this type definition was the first step in writing the parser.

```
type LatexExpr
    = LXString String
    | Comment String
    | Item Int LatexExpression
    | InlineMath String
    | DisplayMath String
    | SMacro String (List LatexExpr)
                    (List LatexExpr) LatexExpr
    | Macro String (List LatexExpr)
                               (List LatexExpr)
    | Environment String (List LatexExpr)
                                      LatexExpr
    | LatexList (List LatexExpr)
    | NewCommand String Int LatexExpr
    | LXError (List (DeadEnd Context Problem))
```

Note that the definition of `LatexExpr` refers to itself, hence is recursive. This is typical of the definitions of types of structured trees. To give an idea of how the parser works, consider the following examples. In each, the first line is source text, the others constitute the resulting AST (line breaks are editorial).

```
> Pythagoras
LXString "Pythagoras"

> \strong{Pythagoras}
Macro "strong" [] [LatexList
                        [LXString "Pythagoras"]]

> \strong{Pythagoras} says that $a^2 + b^2 = c^2$
Macro "strong" [] [LatexList
                        [LXString "Pythagoras"]]
  , LXString "says  that "
  , InlineMath "a^2 + b^2 = c^2"
```

The parser is defined in roughly 500 lines of Elm code and consists of a set of functions which call upon one another. The top-level parser function is given below. Note the close correspondence between its parts and the parts of the type definition. It is built using `oneOf`, a combinator which takes a list of parsers as arguments and which returns a parser as value.

```
latexExpression : LXParser LatexExpr
latexExpression =
    oneOf
        [ texComment
        , displayMathDollar
        , displayMathBrackets
        , inlineMath
        , newcommand
        , macro
        , smacro
        , words
        , lazy (\_ -> environment)
        ]
```

We give one more example, the `macro` parser. It uses the combinators `(|=)` and `(|.)` to *sequence* parsers, thereby forming a new one. The resulting "parser pipeline" operates in the following way. The phrase `|= macro` parses the name of the macro. Then `|= itemList optionalArg` recognizes the list of optional arguments, and `|= itemList arg` does the same for the regular macro arguments. Finally, `|. whitespace` "eats" but ignores whatever white space it finds. The values found by the `(|=)` phrases are taken as arguments of the constructor `Macro` for the `LatexExpr` type, and the result of this function call is a value of type `LatexExpr`.

In this example, `whitespace` is a parser for white space, which can come in different flavors depending on context, e.g., spaces only or spaces and newlines.

```
macro : LXParser () -> LXParser LatexExpr
macro =
    succeed Macro
        |= macroName
        |= itemList optionalArg
        |= itemList arg
        |. whitespace
```

One can continue down the rabbit hole, explaining the parsers `macroName`, `optionalArg`, `arg` and the combinators `itemList`, etc., but we stop here.

What is important to understand is that fundamentally there are only three things in something like the MiniLaTeX parser: primitive parsers, combinators that choose among alternatives, and combinators that sequence other parsers. As a note, eating trailing whitespace is important because in the present setup, lexing and parsing are not separate operations.

4.2 Rendering

The top-level rendering function is much like the top-level parsing function. It analyzes the type of a LatexExpr and dispatches the appropriate renderer, which may in turn call other rendering functions, including the top level one. And so on, down the next rabbit hole of function calls we go. The renderer module constitutes roughly 1300 lines of code.

4.3 Code

Code for the MiniLaTeX compiler as well as the demo app can be found at `github.com/jxxcarlson/meenylatex`. (The strange name is to reserve the name `github.com/jxxcarlson/minilatex` for a future stable version with a polished API.)

The code for the `minilatex.lamdera.app` application is at `https://github.com/jxxcarlson/lamdera-minilatex-app`. All code is open source.

5 Feedback

I am very interested in feedback from the community regarding features, bugs, etc. Of special interest is the subset of LaTeX used: what should it be? Comments to jxxcarlson at gmail.

6 Acknowledgements

I would like to thank Evan Czaplicki, Ilias Van Peer, Mario Rogic, and Luke Westby, all of the Elm community, Davide Cervone, MathJax, and the team at KaTeX.org for their generous and invaluable help. I also wish to thank the Simons Foundation (`https://simonsfoundation.org`) for its support of this project.

This document was originally written in MiniLaTeX and is available at `minilatex.lamdera.app/g/22`.

◇ James Carlson
`jxxcarlson (at) gmail dot com`
`https://minilatex.lamdera.app`

Why the LaTeX community should care about SGML

William F. Hammond

Abstract

Given that the universal format-to-format translator Pandoc is coming of age, LaTeX authors are tempted to think that whatever LaTeX they write can quickly be translated without worry to whatever other format may be required.

Of course, that is not exactly true, but the use of an XML profile of LaTeX can make it exactly true. However, an SGML profile of LaTeX can provide closer emulation of classical LaTeX than an XML profile.

Most actors in the world of markup have restricted their use of SGML to XML. For that reason software that handles SGML beyond the realm of XML seems to be falling out of maintenance. If the LaTeX community wishes to continue to be able to avail itself of the advantages of SGML for LaTeX source emulation, it may fall on the LaTeX community to maintain the extant SGML libraries.

1 Maximally useful source markup

Authors spend a great deal of time writing their articles and reports. Because of that investment of time it is important that tools used in writing are chosen carefully. One wants the fruit of one's writing to be presentable not only on the printed page but also on screens of various sizes from mobile telephones to full-wall monitors. It is better to write once and then use robust processing streams for the desired output formats rather than to edit manually for each output. Within the realm of TeX-like source, the best results flow from profiled LaTeX.

1.1 The concept of LaTeX Profiles

A LaTeX profile is a dialect of LaTeX with a fixed command vocabulary, where all macro expansions must be effective in that vocabulary, having dual existence under an SGML [4] document type with a canonical XML [2] shadow. I spoke about this concept in my talk [7] on the 100000_2-th (i.e., 32nd) anniversary of TUG in 2010.

While an instance of a LaTeX profile may be written directly as SGML or XML using the regular syntax, it is envisioned that one would want to use generalized LaTeX, i.e., write using the syntax of LaTeX.

The XML guise of a LaTeX profile can be styled, though with less than perfection, using CSS [1]. See my talk [8] at TUG 2014. This is not a new idea except insofar as it involves CSS-only rendering of most of LaTeX math.

My GELLMU Project, found at my university website, `albany.edu/~hammond/gellmu`, as well at CTAN (`ctan.org/pkg/gellmu`) provides a didactic model for the use of a LaTeX profile.

1.2 A side remark on accessibility

In reference to the first of Ross Moore's talks at this meeting, I want to suggest that HTML 5 [3], with Unicode text and MathML for math, obtained from a LaTeX profile can rather easily be made accessible for those with vision impairment.

2 Why SGML rather than XML?

The simple answer is that SGML provides better emulation of classical LaTeX than XML in that XML requires markup elements to have tags for both start and end, whereas with SGML it is commonly possible to omit an end tag and less commonly possible to omit a start tag and sometimes both. This is not the place to rehearse the conditions under which these tag omissions are allowed. The point is that classical LaTeX may be viewed as allowing many tag omissions, and for that reason one may construct approximate SGML models that are closer to classical LaTeX than any XML model can be.

One aspect of the overall idea of generalized LaTeX is that the processing should proceed through a pipeline with well-defined stages, and the first stage of that processing should involve only syntax. Thus, for a particular LaTeX markup structure, a human may see how to use code to reformulate it directly under an XML document type, but that formulation might require knowledge of both classical LaTeX vocabulary and the vocabulary of the XML document type.

3 Example: A simple table

There follows a mundane example of a centered table that would normally be created in LaTeX using a *tabular* environment inside a *center* environment.

long phrase	five	shorter
shorter	long phrase	five

The pattern of horizontal alignment in the cells of this table is "rcr", which in LaTeX is normally furnished as an argument of the *tabular* environment. This table has two rows, each with three cells. The rows have horizontal borders and the cells have vertical borders.

In the GELLMU Didactic Production System this centered table can be marked up with

```
\begin{display}
\begin{tabular}{|r|c|r|}
\hline
long phrase & five & shorter \\
\hline
shorter & long phrase & five \\
\hline
\end{tabular}
\end{display}
```

One probably wants the cell separators ("&") to be viewed as tags for cells and the row separators ("\\") to be viewed as tags for rows.

The question here is not how to mark up a table in some corresponding XML but how to formulate XML markup that models this LaTeX construction. Where do the *hline*-s belong in that model? If only because the question about the *hline*-s requires some thought, the translation from this generalized LaTeX to an XML model cannot be just a matter of syntax.

But with a few syntactic conventions, including the recognition of argument syntax on the *tabular* environment to generate a generic "argument" "<ag0>", flagging the \\ as a generic "breaking" element "<brk0>", and recognizing the special syntactic role played in LaTeX by the character "&" leading to the element "tabampcell", one arrives in a straightforward fashion at this segment of SGML:

```
<display>
<tabular><ag0><vbr/>r<vbr
   />c<vbr/>r<vbr/></ag0>
<hline>
long phrase
  <tabampcell>five
  <tabampcell>shorter<brk0>
<hline>
shorter
  <tabampcell>long phrase
  <tabampcell>five<brk0>
<hline>
</tabular>
</display>
```

(In this example "<vbr/>" is an empty element representing the special character "|". In the GELLMU Didactic Production System all 33 of the printable non-alphanumeric ASCII characters have representation as empty elements with three-letter names. There are various context-dependent ways that any of these characters can be special after a format translation. Naming them makes it possible for last processing minute decisions to be made. On the other hand, use of the names is quite often optional in generalized LaTeX markup source, as here with "|", so long as emulation of TeX's *manmac* is not being engaged.)

At the next stage of processing — under an SGML transformation — using code with knowledge of markup vocabulary at both ends, the SGML segment above is transformed to the following XML segment:

```
<display>
 <tabular>
  <tabuhead>
  <tabharg><vbr/>r<vbr/>c<vbr
          />r<vbr/></tabharg>
    <hline/>
  </tabuhead>
  <tabubody>
   <taburow>
    <firstcell>long phrase</firstcell>
    <tabampcell>five</tabampcell>
    <tabampcell>shorter</tabampcell>
   </taburow>
   <taburow>
    <firstcell><hline/>shorter</firstcell>
    <tabampcell>long phrase</tabampcell>
    <tabampcell>five</tabampcell>
   </taburow>
   <taburow>
    <firstcell><hline/></firstcell>
   </taburow>
  </tabubody>
 </tabular>
</display>
```

4 SGML, LaTeX, decline, and authors

Like XML, SGML is a grammar for markup languages. XML has stricter rules than SGML. While formally SGML and XML have disjoint specifications, it is nonetheless the case that any XML document type admitting a "DTD" definition may be realized in a routine way as an SGML document type. In the other direction, most SGML document types admit an SGML normalization that can usually be transformed to an XML document type.[1]

4.1 Why XML now dominates SGML

One of the complications with SGML that led to the rise of XML is that an SGML document very rarely exists as a stand-alone file. An SGML document must always include or reference a formal document type definition and be associated, explicitly or implicitly, with an on-board SGML declaration. As a practical

[1] But, for example, there might be a challenge in this direction with an instance of an SGML document type that makes extensive use of SDATA.

matter an SGML user must have a collection of auxiliary documents just as a LaTeX user must have a collection of packages. While this can be practical for sharing among a group of authors, it makes sharing an SGML document across the web more difficult and less efficient than sharing an XML document.

SGML documents that are not XML have effectively vanished from the web.[2] The fact that any new SGML documents being generated are behind closed doors, together with a somewhat steeper learning curve for SGML than for XML, seems to have led to a loss of interest in SGML beyond XML.

4.2 Who the authors are

How do SGML and XML documents arise?

I believe most of the books and articles written by actual authors, whether for academic publication or for the popular press, are most likely written either with a word processor, such as provided by Microsoft, or in a TeX-family markup.

I believe that most extant SGML or XML documents are not actual source. For example, there are "markdown" languages from which basic XML documents can robustly be spawned. When SGML or XML documents are actual source, the creators are usually persons working, one way or another, in document technology. Those creators usually work with editing tools that have been adapted to minimize the distinctions between SGML and XML that I mentioned earlier in section 2.

I see the LaTeX community as still under challenge by the concern raised by Chris Rowley at the 2010 TUG meeting over "peak TeX" (analogous to "peak oil"), and here I've pointed to SGML (beyond XML) being in decline. One of the threats for the future of LaTeX is its difficulty in being converted to other formats for documents where that is sensible. The concept of a LaTeX profile provides a place where LaTeX and SGML can help each other to the benefit of both.

5 Libraries for parsing and processing

Because the rules for an XML document are somewhat more restrictive than the rules for an SGML document, libraries for processing XML are easier to construct and maintain than libraries for processing SGML. Indeed, the specification of SGML provides for variations of syntax, detailed character set specification, and myriad markup shortcuts to the point that I do not know whether any library was ever produced to provide functions for handling a full implementation of the SGML specification. However, the extent of SGML use that I think desirable for profiling LaTeX is well within the territory covered by SGML libraries.

5.1 Simplicity with XML

With the rise of XML and the ease of writing software for parsing and transforming XML documents, many new options appeared. With XML a document need not be accompanied by a document type definition. It is sufficient that transforming software knows the markup vocabulary used with the document. Thus:

- An XML document may be rendered in a web browser solely by linking the document to a CSS stylesheet.
- An XML document may be transformed to another format by using an XSLT transformation that is defined by creating an XSLT stylesheet, which itself is an XML document.

My guess is that XSLT is probably the most widely used transformation language for XML today. Personally, I find it cumbersome to write for XSLT. I would much rather code in a traditional programming language. In particular, it is not pleasant trying to code for XSLT when the translation target is LaTeX.

5.2 The libraries OpenSP and SGMLSPM

I believe that the most widely deployed library for handling SGML is OpenSP. It is a C++ library for parsing and transforming SGML documents that was spawned from James Clark's SP. Before the rise of XML, I believe Clark's SP was dominant. For example, at some point during the time of Sun Microsystems' Solaris operating system, the system manual pages were re-coded from Roff source to a variant of Docbook SGML, and SP was deployed for generating various output formats. This arrangement for system manual pages is found today in Ubuntu systems though with OpenSP rather than SP. It's not always understood that since every XML document may be construed as an SGML document, OpenSP can be used with XML documents.

I might also mention OpenJade, which was spawned from Jade, also by James Clark. OpenJade provides an engine for *Document Style Semantics and Specification Language (DSSSL)*, which is an early transformation language for SGML that is written in SGML — thus, a forerunner of XSLT — under an SGML declaration that makes one of its stylesheets look like Lisp code. Usually package management

[2] There was a time after the rise of XML that version 4 of HTML, an SGML "application" (document type), was the dominant markup for web pages. It worked because web browsers were required to have native knowledge of HTML. This continued for a while even after the early XML form of HTML was promoted and then appeared to gain widespread use through many web page instances that were not actually well-formed XML.

systems that house OpenSP also house OpenJade; I mention this because online searches for "`openjade`" can be easier than for "`opensp`".

Finally, I want to mention the Perl software SGMLSPM/sgmlspl for SGML transformations written by David Megginson of Ottawa and released as GPL software in 1995 that, to my knowledge — I use it daily[3] — has never since needed repair. SGMLSPM/sgmlspl enables one to write a handler for each element in an SGML (or XML) document type to treat the rendering of that element in a translation. It is designed to accept a parsed stream from OpenSP and generate an output stream in the target format. This works well for writing HTML, LaTeX, and just about any output format. If the handlers are not written mindlessly, a single run of OpenSP piped to SGMLSPM/sgmlspl can handle a very large document. Also there is the advantage that inside one of those handlers, the transformation writer has the full power of Perl.

5.3 OpenSP needs maintenance

Unfortunately, as it is today, OpenSP only supports the basic multilingual plane of Unicode (U+0000–U+FFFF). Tackling the task of adding support for all of Unicode might not at first glance seem hard, but it is daunting because of the complexity of OpenSP that arises from the extent of its coverage of SGML beyond XML and its character handling. To my mind this task might be a good master's thesis project for someone in Computer Science. Prior to modification the code will require much study. The task needs someone with stamina and good eyes.

References

[1] Bert Bos, Tantek Çelik, Ian Hickson, & Håkon Wium Lie, *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*, World Wide Web Consortium Recommendation, 7 June 2011.
`w3.org/TR/2011/REC-CSS2-20110607`

[2] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, & François Yergeau, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, World Wide Web Consortium Recommendation, 26 November 2008.
`w3.org/TR/2008/REC-xml-20081126`

[3] S. Faulkner, A. Eicholz, et al., *HTML 5.2*, World Wide Web Consortium Recommendation, 14 December 2017.
`w3.org/TR/2017/REC-html52-20171214`

[4] Charles F. Goldfarb, *The SGML Handbook*, Clarendon Press, Oxford, 1990.

[5] William F. Hammond, "GELLMU: A Bridge for Authors from LaTeX to XML", *TUGboat* 22:3 (2001), pp. 204–207.
`tug.org/TUGboat/tb22-3/tb72hammond.pdf`

[6] William F. Hammond, "Dual presentation with math from one source using GELLMU", *TUGboat* 28:3 (2007), pp. 306–311.
`tug.org/TUGboat/tb28-3/tb90hammond.pdf`
A video of the presentation at TUG 2007, July 2007, in San Diego is available at `http://zeeba.tv/conferences/tug-2007`.

[7] William F. Hammond, "LaTeX profiles as objects in the category of markup languages", *TUGboat* 31:2 (2010), pp. 240–247.
`tug.org/TUGboat/tb31-2/tb98hammond.pdf`.
A video of the presentation at TUG 2010, June 2010, in San Francisco is available at `http://zeeba.tv/conferences/tug-2010`.

[8] William F. Hammond, "Can LaTeX profiles be rendered adequately with static CSS?", *TUGboat* 35:2 (2014), pp. 212–218.
`tug.org/TUGboat/tb35-2/tb110hammond.pdf`
A video recording of the presentation at TUG 2014, June 2014, in Portland, Oregon is available at `http://zeeba.tv/conferences/text/tex/tug-2014`.

[9] Leslie Lamport, *LaTeX: A Document Preparation System*, 2nd edition, Addison-Wesley, 1994.

[10] MacFarlane, John, *Pandoc: A Universal Document Converter*.
`pandoc.org`

◇ William F. Hammond
University at Albany,
Albany, New York, and
San Diego, California
`whammond (at) albany dot edu`
`https://www.albany.edu/~hammond/`

[3] After incorporating one additional feature written by Dave Walden.

LaTeX technologies at work — aesthetically beautiful PDFs on the fly from XML input: XML Page Composition (XPC) micro-service in the cloud

Rishikesan Nair T., Aravind Rajendran, Rajagopal C.V., Radhakrishnan C.V.

Abstract

XML Page Composition (XPC) is a micro-service in the cloud which is built on the web-based typesetting framework TEXFolio from River Valley Technologies, India, a typesetting technology company established in India in 1996 by the brothers C.V. Radhakrishnan, C.V. Rajendran and C.V. Rajagopal, which acts as a technology provider for STM Document Engineering Pvt Ltd (STMDocs), which in turn uses TEX and friends for typesetting and provides prepress services to leading publishers around the world.

The purpose of XPC is to automate PDF creation from the XML source using an automated workflow without any manual intervention. Of the several recently developed web-based frameworks by River Valley Technologies India, XPC is the newest. Ithal [1], Neptune [2] and TEXFolio [3] are other milestone developments of River Valley Technologies. A few more products and services specifically focussing on empowering the author are under development at River Valley.

A valid XML along with the graphics and other metadata files associated with it should be made available to the XPC system to generate a PDF. An automated quality control (QC) process is performed on the PDF output and a number of parameters, both standard typesetting specifications and publisher-specific requirements, are checked by the system itself as part of the final validation.

1 Introduction

Prepress work for scientific, technical, and medical (STM) journal production has been subjected to enormous changes over recent years, in order to meet growing technology requirements, speed up the production process, and reduce overall production time. The aim is to publish articles as quickly as possible, thus reducing manual labour to increase accuracy and cost reduction. In the beginning, the research articles or other materials were typeset for print media only. However, when the Internet came into the picture the landscape radically changed. The requirement for many different types of outputs become a de facto standard, and the typesetter who does the prepress work has to generate SGML/XML/MathML and web-optimized PDFs in addition to the "fat" PDF or print PDF, all from a single source which the author provides.

The current scenario is that a publisher uses typesetting services either from one prepress supplier, or a few, distributing its journals among them. Those supplier(s) are responsible for the entire production of the particular journal(s) assigned to them:

(1) media conversion,
(2) file structuring,
(3) copyediting,
(4) producing proofs for authors,
(5) incorporating author corrections,
(6) producing XML/MathML, web optimized PDFs, print ready PDFs and electronically publishing them for article-based publishing,
(7) compiling the articles into a journal issue, per the instructions from the publisher.

Leading publishers of STM journals are recently thinking along new lines, trying to distribute the prepress work of even a single journal to many typesetters. For example, steps (1) to (5) to the first supplier; (6) to a second supplier and (7) to a third.

XML Page Composition (XPC), a new product from River Valley Technologies India[1] which STMDocs[2] has evaluated can play a major role in the prepress work industry. XPC is deployed under stage 6 (see above) in a fully automatic mode. Now let us look at XPC in detail.

2 XML Page Composition Service (XPC)

The XPC micro-service is a typesetting system in the cloud to create standards-compliant and aesthetically pleasing PDF using TEXFolio, directly from a valid production XML, assets and metadata. (TEXFolio is the TEX-based typesetting framework in the cloud.)

The Automated Quality Control system (Auto-QC) built into XPC ensures the quality of the generated PDF output and also carries out publisher-specific validation. Auto-QC is based on certain rules and standards which are predefined. Column balancing, float placement, overfull boxes, and underfull boxes are a few of the issues checked by auto-QC. Auto-QC produces an error report in PDF format for the operator.

Currently numerous templates for one of the leading STM publishers are configured. There is no limit on the number of typesetting models that can be configured.

All files will pass through XPC without manual intervention. However, heavy math, depending on

[1] http://www.river-valley.com
[2] https://www.stmdocs.in

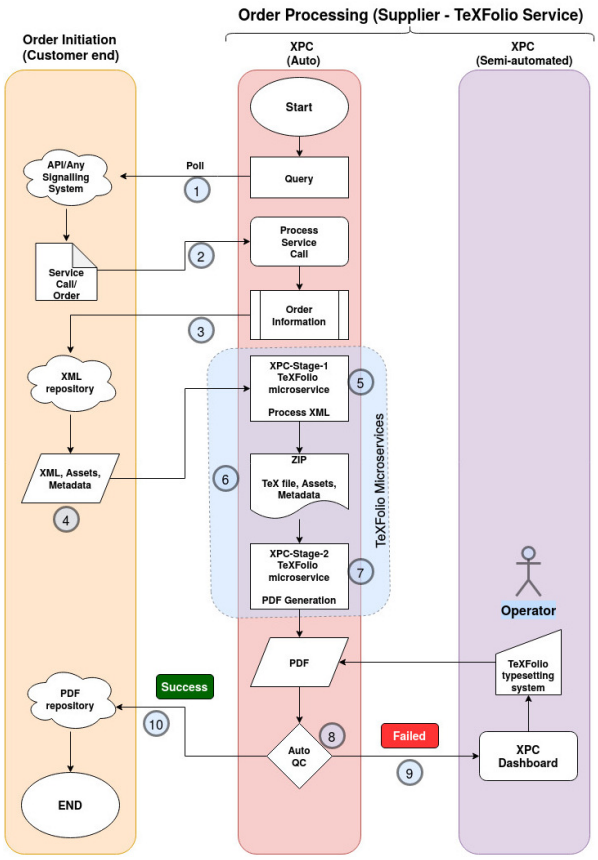


Figure 1: XPC workflow.

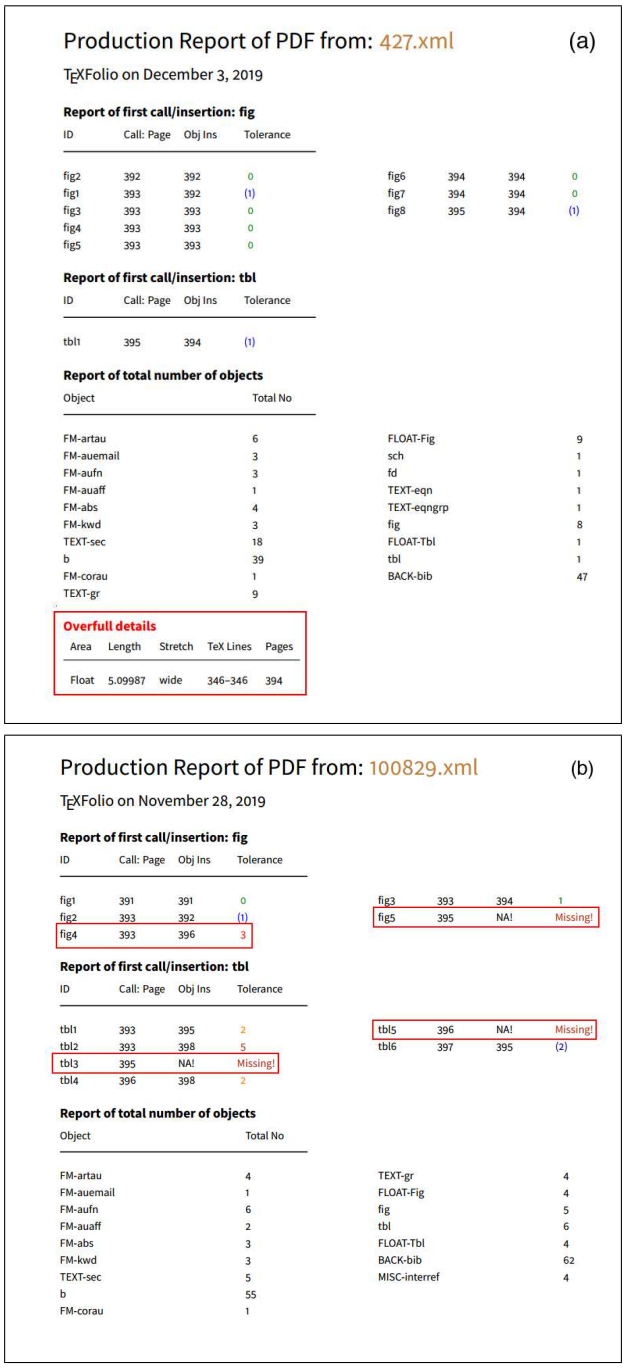


Figure 2: Auto-QC error reports.

the needs of the publisher, may need manual support, mostly with pagination.

2.1 XPC Features

The main features of XPC (the figures are grayscaled for print):

1. XPC workflow (Fig. 1) is in the cloud.
2. No content editing or alterations of the source, hence no accidental human errors.
3. High-level automated QC between PDF output and XML, as described. Two sample reports are shown in Fig. 2.
4. Formatting/pagination of PDF output, if required, is done using a control file generated from the XML, without touching the XML data.
5. Application of artificial intelligence for table formatting and float placement, thereby reducing manual effort.
6. Multilingual support, currently configured for 11 languages.
7. Automatic table width calculation to help typeset tables in either single column or double column mode without any processing instructions.
8. APIs to support command-line operations for automation.

2.2 Workflow

The workflow diagram in Fig. 1 along with the explanation of each step provided in the following will describe in more detail the functionality of the XPC service.

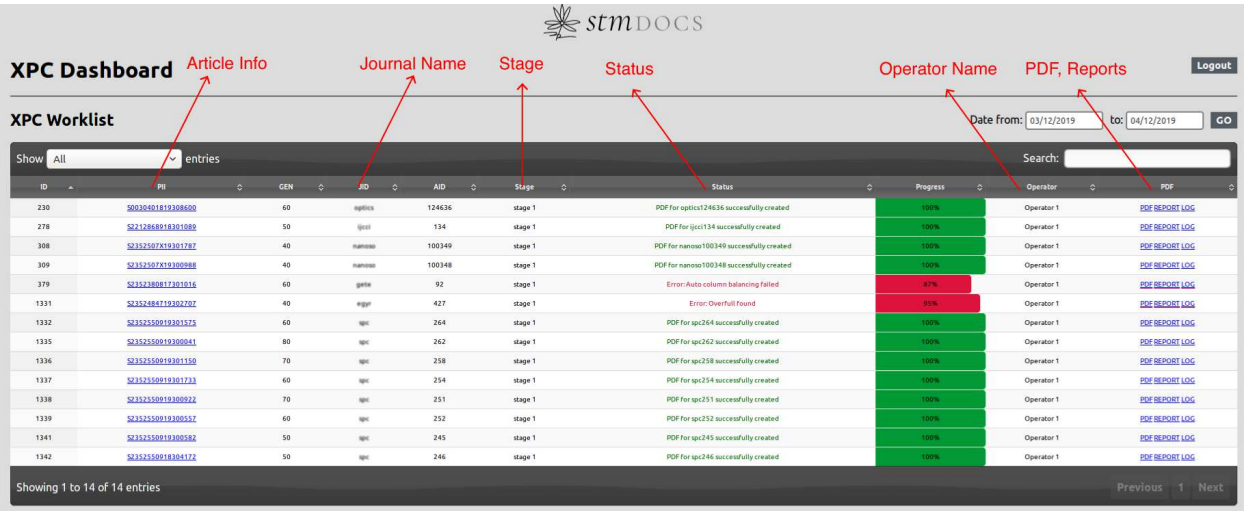


Figure 3: XPC operator dashboard as implemented at STMDocs during evaluation.

1. Order initiation: API service call or check for orders in publishers' servers.
2. Process service call or orders, and extract order information.
3. Send order information to data repository or datastore.
4. Retrieve input XML and assets of the items which are generated by the XML supplier.
5. Process XML: Add namespaces, find table width, create an external float control file and create a TEX file. Find the typesetting model and (only if a journal with a new model is received) create a typesetting template configuration file automatically.
6. Make an archive of all these and push them to TEXFolio microservice for processing.
7. Create PDF.
8. Trigger auto-QC: If the quality of the PDF output is not up to the benchmark or publisher specifications, flag a failure, likely a need for manual pagination.
9. Items requiring manual intervention get listed in the operator's dashboard (see Fig. 3). Operator paginates using the float control file, pushes the finished PDF again for auto-QC.
10. If auto-QC is successful, PDF will be delivered to the client.

2.3 Error reports — Details

Two error reports are given in Fig. 2 (above and below are two separate reports). There are three main sections which will appear in every article which contains figures and tables: "Report of first call/insertion: fig", "Report of first call/insertion: tbl", "Report of total number of objects". The optional section "Overfull details" will appear only if the PDF output has any overfull text.

One of the many challenges of the auto-pagination function is the placement of floats near their references. XPC will do a fairly nice job here, however in very rare cases due to severe constraints such as a small number of pages, a large number of floats, and a two-column document, as one can imagine, it is a difficult task to place the floats near to their references even manually. If the floats are placed far from their citations, this information will be flagged in the report and the operator who checks the report can find and correct it. As you can see in the sample reports, the following details are included to help the operator to find the problem:

- `ID`: The ID of the float to search.
- `Call Page`: The page in the PDF where the float is first cited.
- `Obj Ins`: The PDF page where the float is inserted.
- `Tolerance`: The tolerance with which this can be allowed. As the tolerances become worse, they are highlighted with colour changes, red being the worst.

2.4 Issues and challenges

The developers faced many challenges during the development of XPC. A few of them are listed here:

1. Finding journal typesetting model
2. Table cell width calculation
3. Float placement
4. Handling built-up accents

5. Pagination using a control file automatically generated from XML
6. Automated QC of the PDF output
7. Automated column balancing of the last page in a two-column article
8. Automating manual fall-out: Adding more pagination commands in control file — *In Progress*

All except the last have been resolved.

2.5 Estimated production capacity

The service levels presented as part of the pilot phase of the service were these:

1. 30% of the articles can be delivered within 6 hours.
2. 60% of the articles can be delivered within 12 hours.
3. 100% of the articles can be delivered within 24 hours.
4. Capacity that can be handled would be 250 articles per day (average 15 pages/article).

However the performance has been greatly enhanced after the pilot phase. Currently the service capacity is over 600 articles per day (average 15 pages/article).

3 Acknowledgement

STM Document Engineering (STMDocs) gratefully acknowledges and thanks River Valley Technologies for allowing their products and technologies (XPC, TEXFolio, NEPTUNE, Ithal, etc.) to be showcased at TUG. Any credits are to be attributed to River Valley Technologies. We also highly acknowledge the team at STMDocs who have helped with the evaluation of the XPC system, especially Apu V and Rahul Krishnan S and our testing team Akshay K.S. and Sangeetha V.

References

[1] Ithal. https://ithal.io/main.html

[2] R. Aravind Rajendran, Rishikesan Nair T. NEPTUNE — a proofing framework for LaTeX authors. *TUGboat* 40(2):150–152, 2019. https://tug.org/TUGboat/tb40-2/tb125rajendran-neptune.pdf

[3] R. Rishikesan Nair T., Rajagopal C.V. TEXFolio — a framework to typeset XML documents using TEX. *TUGboat* 40(2):147–149, 2019. https://tug.org/TUGboat/tb40-2/tb125rishi-texfolio.pdf

◇ Rishikesan Nair T.
Aravind Rajendran
STM Document Engineering Pvt. Ltd.
River Valley Campus, Mepukada
Malayinkil
Trivandrum 695571
India
rishi (at) stmdocs.in,
aravind (at) stmdocs.in
https://stmdocs.com

◇ Rajagopal C.V.
Radhakrishnan C.V.
JWRA 34
Jagathy
Trivandrum 695571
India
cvr3 (at) river-valley.org,
cvr (at) river-valley.org
http://river-valley.org

Tagging with LaTeX — Part 1: Author considerations

Ross Moore

Abstract

Successful tagging within PDF files generated from LaTeX source encourages a change in viewpoint on the nature and intent of the LaTeX coding. Using explicit examples from a real-world document, we illustrate how to capture such a change within the LaTeX source, for various structural elements. Other issues for creating archival and accessible PDF documents are discussed.

1 Introduction

With 'Tagged PDF' being the accepted method for creating PDF documents enriched to satisfy Accessibility requirements [5, 6], this article is intended to address the main issues that authors and editors should be aware of, with regards to tagging and LaTeX usage. Examples are taken from a real-world fully-tagged research report, prepared in LaTeX but also employing extra coding written by the author, in a package named `tpdf` that handles the technical aspects of producing 'Tagged PDF'. That research report is the one used by the author in the talk [7] at TUG 2019 (delivered remotely using Zoom collaboration software). It is based on a publication from the U.S. National Parks Service [8].

This article is not meant to be an introduction to the use of the `tpdf` package, but more about the kind of extra considerations that authors and editors alike should be making, to allow tagging to be performed successfully and usefully.

1.1 Requirements for U.S. federally funded research publications

As some justification as to why 'Tagged PDF' is both relevant and desirable, we note some U.S. Government guidelines and requirements.

- United States Access Board; Information and Communication Technology (ICT) Final Standards and Guidelines. [1]
  Section 508 ICT Refresh, §504.2.2:

  > ... be capable of exporting PDF files that conform to ANSI/AIIM/ISO 14289 -1:2016 (PDF/UA-1) ...

- National Science Foundation, Q&A: public access policy. [2]

  > ... possess a minimum set of machine-readable metadata elements ...;
  > be managed to ensure long-term preservation; ...

By producing documents conforming to published standards, both PDF/A [3] and PDF/UA [4], these obligations can be met in full, if not surpassed.

2 Tagging commands for special content

It is a common typographical practice to use different styling to present names of books, magazines, or publications which have a particular relevance to the topic under discussion. Certainly the fact of a different style being used indicates to a fully-sighted reader that there is a special significance, but not what that significance actually is. That has to be deduced from context. With tagging, that significance can be made explicit. And with LaTeX source this is very easy to do.

For example, the Night Skies Project report has a page which mentions the various Acts of Congress which underpin their work; see Figure 1. The names of these acts appear in italics. This was originally done by simply specifying

```
\textit{Organic Act of 1916} ...
```

Changing this by inventing a macro `\nrpsAct` the true intention is captured, at least within the LaTeX source. For typesetting purposes the expansion of this new macro is given by:

```
% for names of Acts of Congress
\newcommand{\nrpsAct}[1]{\textit{#1}}
```

which typesets exactly as `\textit` does. But now, when it comes to generating tagging for a Tagged PDF version, as seen on the left-hand side of Figure 1, one can use coding as in Figure 2.

In that coding we see that firstly a new macro is created, named `\NRPS@Act`, which refers to the same code-block as currently does `\nrpsAct`, by using TeX's `\let` primitive. Then a new code-block is defined under the name `\TPDF@NRPS@Act`. When this block is executed, after reading the parameter text as `#1` a group is started with `\begingroup`. Structure tagging named as `CongressAct` is implemented, along with a counter for such structure elements. The tagging that would otherwise be performed by `\textit`, through the style change macro `\itshape`, is suppressed since we are using the `CongressAct` structure instead. Then `\TPDF@NRPS@Act` is called with the `#1` parameter text, to do the typesetting and generate the associated content tagging structures that would normally have been done when `\textit` is called. The grouping is closed using `\endgroup` after which normal tagging of the paragraph content is resumed. To have this new code block activated at the correct time, we re-assign the macro name `\nrpsAct` to point to the modified expansion, again using `\let`. A final command, using

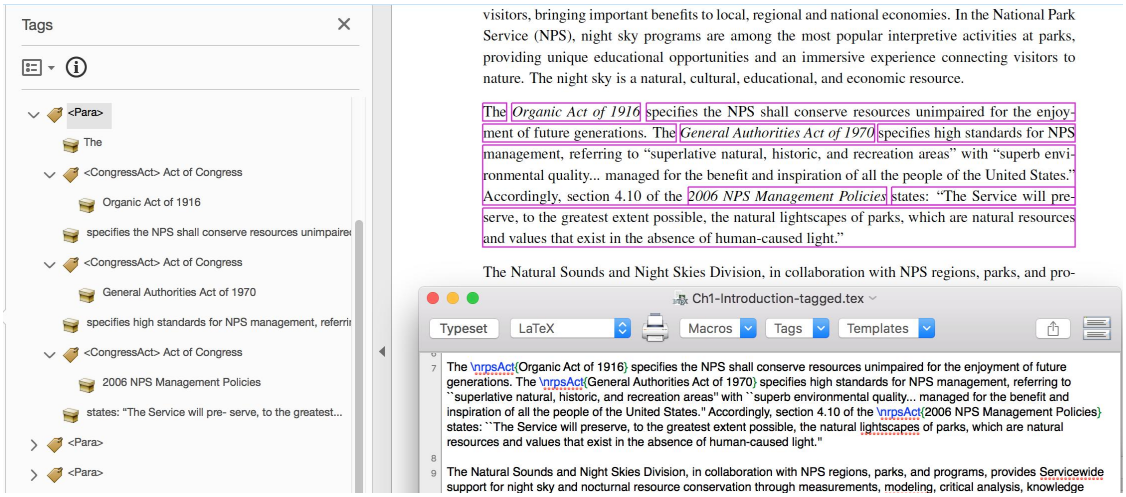


Figure 1: Tagging of Acts of Congress, using a distinctive macro name `\nrpsAct`.

```
\let\NRPS@Act\nrpsAct
\def\TPDF@NRPS@Act #1{%
 \begingroup
  \TPDF@advancecounter{CongressAct}%
  \edef\TPDF@theseparams{{CongressAct}{CongressAct.\TPDF@counter@CongressAct}}%
  \expandafter\TPDF@newstructnode\TPDF@theseparams
   {}{}{}{Act of Congress}{}{}{}%
  \TPDF@style@structure@suppress{\itshape}%
  \NRPS@Act{#1}%
 \endgroup
}
\let\nrpsAct\TPDF@NRPS@Act
\TPDF@appendto@RoleMapDict{/CongressAct /Span}
```

Figure 2: Code to initiate generation of the structure tagging as seen in Figure 1.

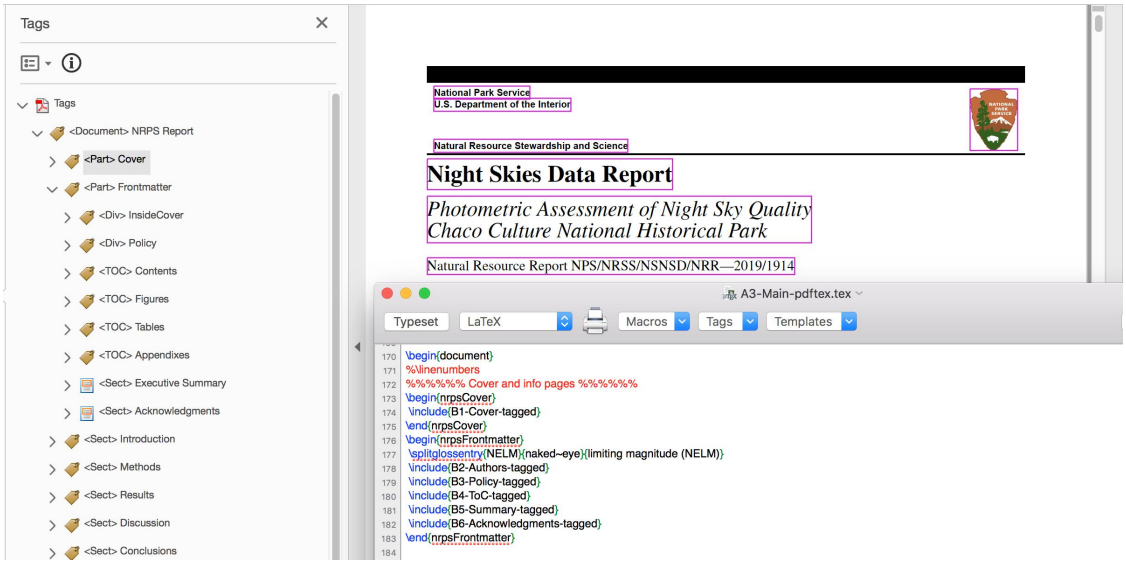


Figure 3: Structure tagging of the front cover and front-matter section of the research report.

`\TPDF@appendto@RoleMapDict`, allows PDF reader software to treat the non-standard structure tag name `/CongressAct` (which we just invented) in the same way as the standard `/Span` tag name.

2.1 Aside on how TEX macros work

We all make great use of (LA)TEX commands, but how well do we understand how they work? Upon reading a string beginning with the \ character, TEX (the program which underlies LATEX processing) creates a control-sequence token (also called a 'macro'); but what is it really? Essentially there is the name token (such as `\title`, `\date`, `\section`, etc.) and a block of coding that is to be executed, or data to be inserted into the processing stream, when the name token is encountered while working through a document's source. This block of coding resides somewhere in the computer's memory, associated with the running job. The memory location has an address, specified by a number which allows the code to be found and used. In other computing languages, one refers to use of such address locations as *pointers* to data or code-blocks. Thus effectively a macro is just a *named pointer* to a block of code or content to be used.

When a new macro is first defined (e.g., using LATEX's `\newcommand` or similar, or with TEX's `\def`, `\edef` or `\let`) one can think of a key–value pair consisting of the new name token and its associated memory address, being pushed onto the top of a stack of all the currently defined such name tokens. Now when a command is encountered, the processor starts at the top of this stack, reading downwards until a token having the same name is encountered. Then the corresponding address is used to find the data or code that is to be handled next. Frequently the macro definition will occur within an environment or grouping (using `{...}`, or `\begingroup` ... `\endgroup` or similar). Upon entering the grouping, the current location of the stack is recorded as a *stack-pointer*, say. When the grouping closes, any name token entries added later than that recorded level are discarded — except for any that have been declared as *global* (using TEX's `\gdef`, or `\xdef` or `\global\let`). All of LATEX's counters are globally defined and updated. This is the kind of mechanism that allows the same macro name to refer to different values, at various stages of processing; a concept known as *scoping* of variable values.

With this interpretation we can better understand how the coding in Figure 2 works. Firstly a new pointer named `\NRPS@Act` is made, pointing to the value of `\nrpsAct`, since that is going to be redefined to point instead to the coding of `\TPDF@NRPS@Act`. But part of the coding of this is to use `\NRPS@Act` itself. This is done within a grouping, because the command

```
\TPDF@style@structure@suppress
```

makes some changes to other macros, and these changes need to be scoped to within that grouping only. In particular a command

```
\TPDF@maybe@taggedparagraphmiddle
```

is set to `\relax`, (i.e., to do nothing) and

```
\aftergroup
 \TPDF@maybe@taggedparagraphmiddle
```

issued. This causes the coding for resumption of tagging within the surrounding paragraph to be delayed until `\endgroup`, when otherwise it would have occurred upon completion of `\textit{...}`. It works since the named pointer's associated value reverts back to what it was prior to `\begingroup`, as the pointer to the `\relax` value has been removed from the stack.

2.2 Patching as 'hacking' or 'enhancement'

This technique, of capturing a pointer and using it within a new code-block, to replace what LATEX would do evaluating a particular macro name, can be considered as a trick for code-hacking. Or it can be considered as a legitimate technique for enhancing the results of typesetting with other structures that may be important for the job as a whole. Such patching is used in the nameref package, part of the hyperref bundle of packages, for enhancing LATEX commands to produce named destinations to act as anchors for hyperlinks to section titles, figure/table captions, and other usages of the `\label` command. Other packages employ this technique to enhance footnotes, cross-references, citations and more, with hypertext features.

Later, in Section 6.1, we give an example of how pointers can be used to resolve an apparent inconsistency created by the same LATEX macro being patched by two different packages [20, 21]. Then, in Section 7.2 we use them to resolve a difficulty with mathematical source initially intended for LuaTEX. Furthermore these are fundamental to how the tpdf package works, to produce Tagged PDF output; as will be discussed in more detail in a later paper in this series.

3 LATEX environments indicate structure

Common LATEX environments, such as `flushleft`, `flushright`, `center`, `quotation`, etc. adjust the way paragraph content is displayed. This is a visual hint to a sighted reader that there is a special meaning attached to the enclosed content. But that is exactly one of the main reasons for tagging, to attach a

name to a block of content. That is, a screen-reader will not do anything special with how the content is presented, but it can alert a visually-impaired user to the fact that it has been tagged specially. The `tpdf` package, when used to create Tagged PDF, automatically produces the appropriate tagging for such environments. It also specifies suitable attributes which hint at the desired visual layout, for tools that need to construct a layout; e.g., for small screens.

Other environments, `tabular`, `verbatim` and common list-like environments `itemize`, `enumerate`, `description` and more, indicate that their content may involve special typesetting and/or layouts. For the list-like environments, in particular, the question arises as to whether the listed information remains as part of the preceding (perhaps surrounding) paragraph, or whether it constitutes a separate (vertical) block with the paragraph having finished already. LaTeX has no formal way to differentiate between these cases, however when coded such as follows with a blank line,

```
... satisfying conditions in this list.

\begin{enumerate}
 \item
```

one would expect the paragraph to finish prior to commencement of the list, as triggered by the blank line. Whereas with the following coding

```
... satisfying the conditions:
%
\begin{enumerate}
 \item
```

there is no (uncommented) blank line; indeed the list items might each be supplying a possible end to the preceding incomplete sentence. A commented blank line is only for clarity in the LaTeX source; it can be omitted altogether, with no change to processing.

3.1 New environments declaring structure

Just as new commands can be defined to convey the intention associated with styling, so also can new environments be defined, doing no extra typesetting, but conveying a name to be associated with definite structure within the document. The first two pages of the Night Skies Project report are meant to be printed on special stiffer paper, and serve as the 'cover' for a printed version. Similarly the last two pages are for the back cover, printed on the same stiff paper. Thus in the PDF, it makes sense to regard these 2-page blocks as separate parts of a structured document, tagged as `Part`.

Figure 3 shows an environment `nrpsCover`, that wraps the first two pages, but otherwise produces no visual content. Similarly there is an environment `nrpsFrontmatter` which encloses the title-page, the Table of Contents, List of Figures, Lists of Tables and Appendices, and further information generic to the organisation of the project, rather than being unique to the project report and results. These two structures do not use a separate sectioning command, but they are sufficiently important that a *bookmark* is appropriate within an electronic document.

Coding as shown in Figure 4 achieves this, using a macro `\pseudochapter` which itself uses macros from other packages. The `\NR@schapter` is a patch of `\@schapter` resulting from `\chapter*` for starting an unnumbered chapter. But there is no anchor text, so the prior redefinition

```
\renewcommand{\H@old@schapter}[1]{}%
```

causes that part to be skipped, leaving just the specification of a named destination. Then we have

```
\addbookmarksline{chapter}{#1}
```

to add a bookmark without also creating a Table-of-Contents entry. Its definition, shown in Figure 4, calls up `\addcontentsline`, but with a disabled `\addtocontents` command, which will just gobble its arguments.

These two new commands `\pseudochapter` and `\addbookmarksline` are essentially patching macros `\NR@schapter` and `\addcontentsline` respectively; but this time *removing* functionality, rather than adding extra coding. The result is an invisible chapter heading with a bookmark but no ToC entry. This also provides a place for structure to be attached, as shown in Figure 3.

3.2 Structure destinations

Observe in Figure 5 that the bookmarks use the icon shown here at right below. This indicates that the bookmark includes a specified *structure destination* (see [14, Table 8.4] or [16, Table 151]), as well as the usual destination area of a printed page.

normal bookmark

with structure destination

In the visual view there is no noticeable difference in the result upon clicking on a bookmark. But when the PDF file is exported to XML from Acrobat Pro DC [17], then targets and links are automatically produced for each bookmark; *viz.*

```
<bookmark title="Introduction">
<destination structID="LinkTarget_591"/>
```

```
% declare some environments, to allow hooks for structural tagging
\newenvironment{nrpsCover}{%
 \pseudochapter{Cover}%
}{\unskip}

\newenvironment{nrpsFrontmatter}{%
 \pseudochapter{Titlepage}%
}{\unskip}

%%  RRM:  use this to get a bookmark, at the  chapter level
%%   without making a ToC entry
\newcommand{\pseudochapter}[1]{%
 \begingroup
  \renewcommand{\H@old@schapter}[1]{}%
  \NR@schapter{#1}%
  \addbookmarksline{chapter}{#1}%
 \endgroup
}
\newcommand{\addbookmarksline}[3][toc]{%
 {\renewcommand{\addtocontents}[2]{}%
  \addcontentsline{#1}{#2}{#3}%
 }%
}
```

Figure 4: Code for environments named for the structure tagging as seen in Figure 3, and the extra bookmarks as in Figure 5.

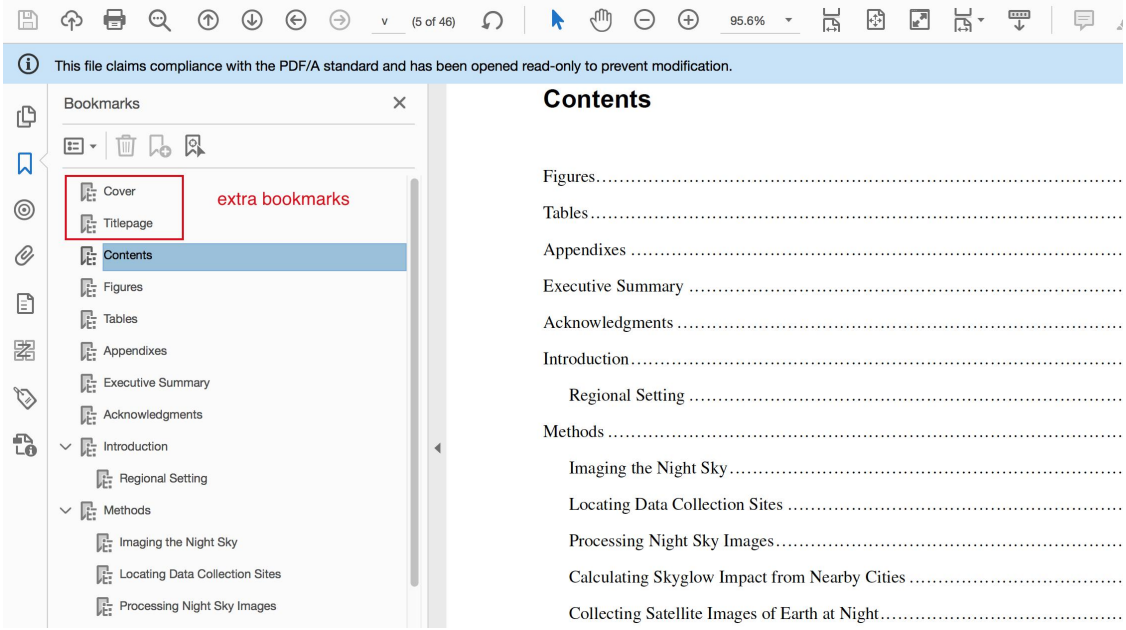


Figure 5: Extra Bookmarks generated for front cover and title-page of the research report, resulting from the coding shown in Figure 4.

```
<bookmark title="Regional Setting">
<destination structID="LinkTarget_622"/>
</bookmark>
</bookmark>
  ...
<Chap>
<H2 id="LinkTarget_591" >Introduction</H2>
 ...
<Sect>
<H3 id="LinkTarget_622" >Regional Setting</H3>
 ...
```

With a *structure destination* the target is the first structure having textual content, as a (perhaps) sub-structure child of a structure element specified in the bookmark's dictionary, or indeed the specified structure itself. Usually this will be the `<H2>` or `<H3>` title text for a chapter or section rather than their parent `<Chap>` or `<Sect>` structures, but it can be different. For example the '`Cover`' bookmark in Figure 5 goes to the first piece of text on the Cover page, namely '**National Park Service**', as seen in Figure 3.

Having a *structure destination* as just described, when exported for other technologies (in particular Assistive Technology for screen-readers, etc.), a bookmark now behaves as would be expected in XML or HTML web-based pages, say. On the other hand, when using 'ordinary' bookmarks, Acrobat Pro DC [17] uses a heuristic to try the deduce the best piece of text to be chosen as the target, given the area of the page specified by its destination key. While working well in most cases, this can come earlier than one would expect from the visual view, and the same target may be found for multiple bookmarks, which is clearly incorrect.

This concept of *structure destination* can be used for hyperlinks as cross-references, ToC entries, citations, etc., at least with PDF 2.0 [16, §12.3.2.3 and Table 202]. These must be specified explicitly, by adding XML attributes directly to the PDF dictionary for the structure elements of both the link and its target. Unless it is also a target for a bookmark, the target needs to be associated with an explicit `id="..."`, with `...` replaced by a unique name. This is achieved with `/id (...)` as a key–value entry, while the link structure needs `xlink:target="..."`, and include also a namespace declaration as follows:

```
xmlns:xlink="http://www.w3.org/1999/xlink"
```

This latter is best done using a `ClassMap` entry for the structure, as `/C /XLink`. While structure destinations were only introduced with PDF 2.0 [16], by including the `/SD` key with value, as well as an ordinary `/D` key and value, in a *go-to action* (`/A`) dictionary, one achieves both forward- and backward-compatibility with all versions of PDF for internal hyperlinks within the document. Reader software is supposed to respect the `/SD` action in preference to the `/D`, if able to do so. Otherwise, as with older software, it will be ignored and `/D` used.

4 Combining environments and commands

As well as the first two pages being cover-page material to be printed on special paper stock, so also are the last two pages. Figure 6 shows the result, with special tagging of the content appearing on the back cover page, using specially defined commands `\nrpsService`, `\nrpsDepartment` named to convey the intention of the material in their arguments, as described in Section 2. Also `\nrpsPlaceLogo`, and specially named environments are used.

The original coding for those last two pages is given in Figure 7, which can be seen as a quite complicated mixture of styling and layout commands, interspersed with the content to be displayed. That kind of coding has been simplified by absorption into macro and environment definitions as shown in Figure 8. Note that `\nrpsIssue` is Metadata that identifies the particular publication, so should be set at the beginning of the document source where such data is easily found; e.g., by (in this case):

```
%% MetaData that is re-used
\providecommand{\theissue}{310/152635}
\providecommand{\theissueII}{\theyear/1914}
\providecommand{\theyear}{2019}
\providecommand{\thedate}{April \theyear}
\providecommand{\thejournal}{Natural
 Resource Report NPS/NRSS/NSNSD/NRR}
\edef\nrpsIssue{\theissue, \thedate}
```

Use of `\providecommand` is recommended for these, since this has an effect only if the macro is currently undefined. This means that it does not interfere with a more sophisticated workflow in which values for `\theissue`, `\thejournal`, `\thedate`, etc. may have been supplied already. The document's title and subtitle could also be provided this way. However, in practice these are set at different sizes in different places within the document, which can require some extra markup to create the best visual layout. So we do not do this here.

Another aspect visible in Figure 6 is the use of a tabular environment simply for the purpose of visual layout. There is no semantic meaning attached to this environment usage, so there is no associated structure tagging; whereas the content of each cell does have semantic meaning, associated with the specially defined macro names, as mentioned above. This is an important counter-example to the discussion of Section 3. Thus within a tagging

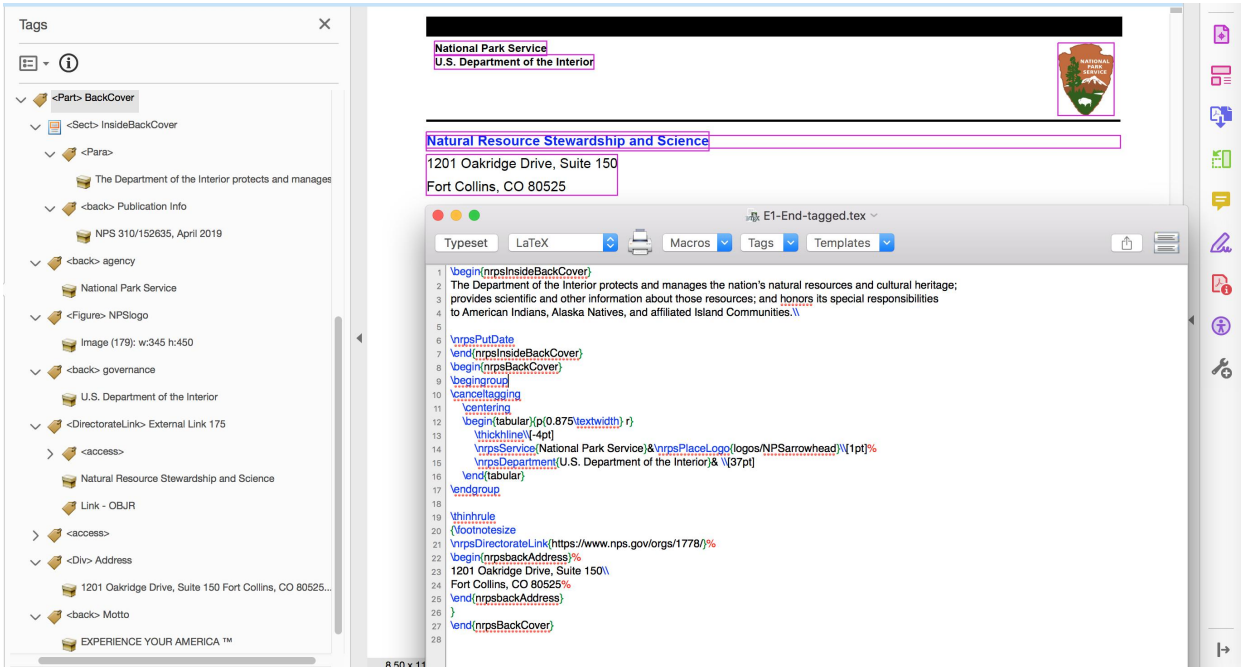


Figure 6: Back cover pages, and the coding for its structures.

```
\clearpage
\pagestyle{empty}
\strut
\vfill
The Department of the Interior protects and manages the nation's natural resources and cultural heritage;
provides scientific and other information about those resources;
and honors its special responsibilities to American Indians, Alaska Natives, and affiliated Island Communities.
\\

NPS \hl{XXXXXX}, March 2019

\clearpage
\newgeometry{lmargin=0.75in, rmargin=0.75in, tmargin=0.75in, bmargin=1in}
\renewcommand{\arraystretch}{0.65} % Default value: 1
\begin{table}
    \centering
    \begin{tabular}{p{0.875\textwidth} r}
        \thickhline
        \\[-4pt]
        \scriptsize{\textsf{\textbf{National Park Service}}}
   & \multirow{3}{*}{\includegraphics[width=0.075\textwidth]{logos/NPSarrowhead}}\\[1pt]
        \scriptsize{\textsf{\textbf{U.S. Department of the Interior}}}     & \\[5pt]
        %\noalign{\hrule height 1.2pt}
    \end{tabular}
\end{table}

\hrule height 1.2pt
%\vspace{4pt}
\footnotesize{
\textsf{\href{https://www.nps.gov/orgs/1778/}{\color{blue}
 \textbf{Natural Resource Stewardship and Science}}}\\
\textsf{1201 Oakridge Drive, Suite 150}\\
\textsf{Fort Collins, CO 80525}\\
}

\vfill
\textsf{\textbf{EXPERIENCE YOUR AMERICA $\rm^{\textsf{TM}}$}}
```

Figure 7: Original coding for the content displayed on the Back-Cover pages.

```
\newcommand{\thickhline}{\noalign{\hrule height 14.15pt}}
\newcommand{\thinhline}{\noalign{\hrule height 1.2pt}}
\newcommand{\thinhrule}{\hrule height 1.2pt}

\def\nrpsPutDate{\noindent\nrpsTheDate}
\def\nrpsTheDate{NPS \nrpsIssue}
\def\nrpsDirectorateName{Natural Resource Stewardship and Science}

\newcommand{\nrpsService}[1]{\scriptsize{\textsf{\textbf{#1}}}}
\newcommand{\nrpsDepartment}[1]{\scriptsize{\textsf{\textbf{#1}}}}
\newcommand{\nrpsDirectorate}[1]{\scriptsize{\textsf{\textbf{#1}}}}
\newcommand{\nrpsDirectorateLink}[1]{{\footnotesize\bfseries\sffamily\color{blue}%
 \href{#1}{{\nrpsDirectorateName}}}}
\newcommand{\nrpsPlaceLogo}[1]{\multirow{3}{*}{\includegraphics[width=0.08\textwidth]{#1}}}

\newenvironment{nrpsbackAddress}{\\\noindent\sffamily\ignorespaces}{\unskip}%
\newenvironment{nrpsInsideBackCover}{\clearpage \strut \thispagestyle{empty}\vfill}{}
\newenvironment{nrpsBackCover}{\clearpage
 \newgeometry{lmargin=0.75in, rmargin=0.75in, tmargin=0.75in, bmargin=1in}
 \renewcommand{\arraystretch}{0.65}% Default value: 1
 \thispagestyle{empty}}%
 {\vfill \begin{nrpsMotto}
  EXPERIENCE YOUR AMERICA\texttrademark
 \end{nrpsMotto}}
\newenvironment{nrpsMotto}{\sffamily\bfseries}{}
```

Figure 8: Macro and environment definitions for styling the content on the Back-Cover pages.

context, as with the tpdf package, there is no need for automatic tagging at this point; hence the use of `\canceltagging`, which affects also the styling commands in the macro expansions; these revert to having just their usual LaTeX expansions. In a future version, this could be incorporated into the tagging expansion of the nrpsBackCover environment, as there is really nothing else there that is associated with implicit semantic macros or environments. It is not done in this article to illustrate the adaptability of tagging to particular semantic requirements within a specific document or class of documents. To allow the document to be processed *without* having the tpdf package loaded, use coding such as follows.

```
\makeatletter
\AtBeginDocument{%
 \@ifpackageloaded{tpdf}{}%
  {% coding to cancel commands
   \let\canceltagging\relax
  }%
}% end of \AtBeginDocument
\makeatother
```

Use of `\AtBeginDocument` delays testing in case the package is loaded later within the preamble section of the document source.

Also observe in Figures 6 and 7 that the original use of `\begin{table} ... \end{table}` has become simply `\begingroup ... \endgroup`, since there is no intention of this material floating elsewhere, and there is no need for a caption or any numbering or other associated constructs.

This use of a tabular environment to control the visual layout occurs also on the front cover, as can be seen in Figure 3. Indeed the coding is identical apart from how the `\nrpsDirectorateText` is displayed, as the name 'Natural Resource Stewardship and Science' of the Directorate [9]. On the front cover, this is part of the tabular, just above a thin rule that finishes this material. On the back cover it appears in larger type below the rule, as anchor text to the website of the Directorate. With this double usage, the structure is declared specially as `/DirectorateLink`, which uses a `RoleMap` entry to exhibit the behaviour of a `/Link`. The coding shown in Figure 8 has macro definitions for both the front- and back-cover instances.

5 'Alternative text' for figures

Accessibility guidelines require that figures be accompanied by 'alternative text' [11] that can help a visually or cognitively disabled person understand the semantic content associated with the inclusion of a figure. By a 'figure' here, we mean non-textual content that has a definite semantic meaning within the context of the electronic document. The alternative text is read by screen readers, and other Assistive

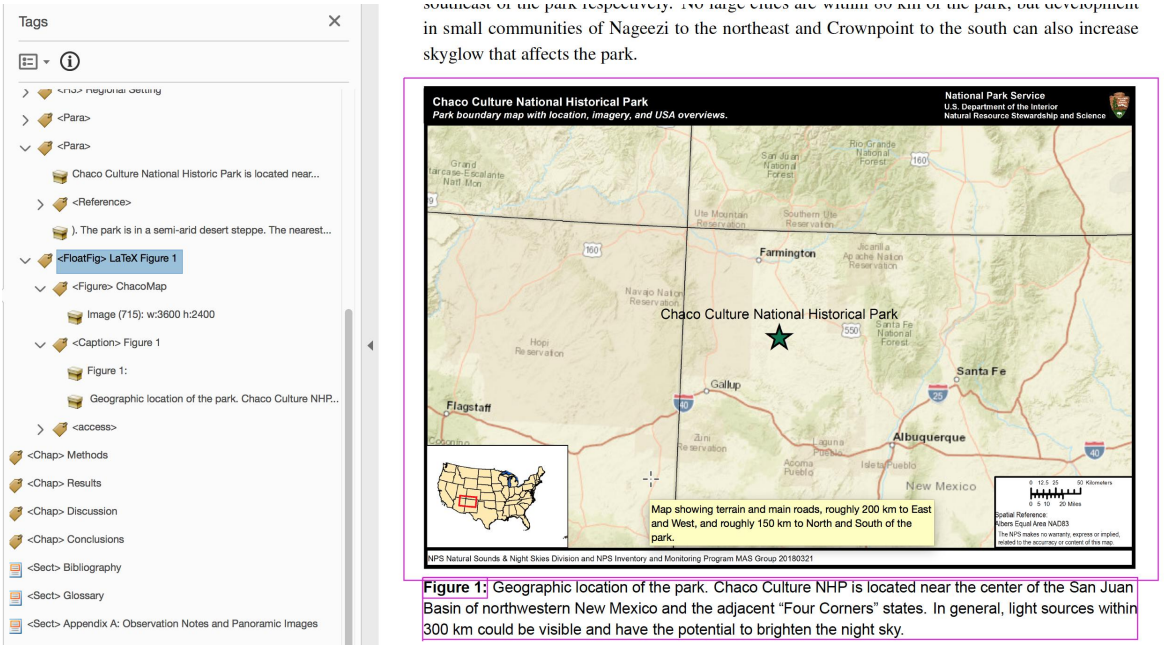


Figure 9: A 'floating' figure with numbered caption following immediately after the image.

technology, in place of the figure itself, which may be a photograph or other image, but need not necessarily be so.

Note that this is logically different from LaTeX's figure environment, which is semantically a grouped block of content within the document which needs to be shown together. For pagination considerations this group, or `/Div`, may need to 'float' to a page later than where the information might otherwise logically be read, or an image first viewed. Usually there will be a caption which describes some of the semantic meaning of the non-textual content, which is often an image or graphic, but need not always be. For example, within this article many of the 'figures' are code listings, which in a Tagged PDF document would be tagged as either `/Code` or `/BlockQuote`, or something equivalent.

A common misconception is that the alternative text for an image is just the same as the 'caption' of a figure. This is quite wrong, especially as any caption should also be available to the Assistive Technology (see Recommendation [6, §4.3.1.3]). Rather the 'alternative text' should complement any discussion in the surrounding text, which could well include a caption. Thus it can explain the 'What?' (is in the image/figure) while the caption is giving the 'Why?' (is an image used here). See Figure 9 for a typical example, where the alternative text is seen in a popup as the mouse hovers over the image. The automatically-generated 'Figure 1:' is in a separate text container to the bulk of the caption.

See [6, §4.3.1] for a discussion and examples of tagging figures and captions, where the main expectation is for a `/Caption` tag to follow immediately after the `/Figure`. However, it is also noted there [6, §4.3.1.1, Example D] that an actual `/Caption` need not always occur.

As a first example of how a non-captioned figure can be implemented, consider the logo of the National Parks Service, as displayed on the front cover and seen in Figure 3, and also on the back cover in Figure 6. When building the Tagged PDF version of the report the `\nrpsPlaceLogo` macro is replaced by an alternative, declared as follows.

```
\def\TPDF@nrpsPlaceLogo #1{%
 \multirow{3}{*}{\tagFigure[NPSlogo]{%
  National Park Service logo, in shape of
  stone arrowhead with Sequoia tree, bison,
  lake and mountain scenery}{%
 \includegraphics[width=0.08\textwidth]%
 {#1}}}}
```

Here the `\tagFigure` macro is defined in the `tpdf` package; it has an argument for the alternative text, as well as an optional name for the figure, and final argument for placing the figure content itself; in this case as an included image of specified width.

As there is no caption, the alternative text is providing a brief answer to the question 'What is the content conveyed by the image?' [12]. In the logo,

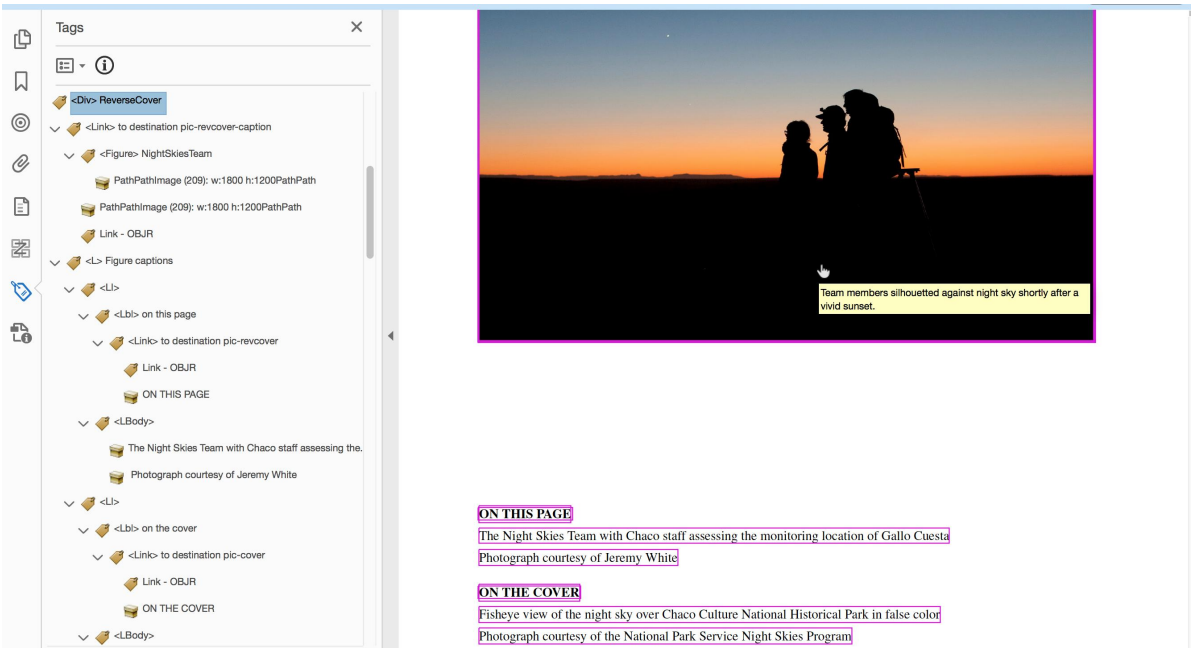


Figure 10: Photo appearing on reverse side of the cover page, along with captions and attributions, for that image and the one appearing on the cover page itself. The image is the anchor for a hyperlink to the caption.

```
\begin{nrpsReverseCover}

% Second page photo
\begingroup
 \hypertarget{pic-revcover}{}% creates anchor-point above the image
 \centerline{\hyperlink{pic-revcover-caption}{%
  \tagFigure[NightSkiesTeam]{Team members silhouetted against night sky shortly after
   a vivid sunset.}%
  {\frame{\includegraphics[width=\textwidth]{photos/fig02-cover2.jpg}}%
     }}% end of  \hyperlink
 }% end of  \centerline
\endgroup
\vfill

% Figure captions
\begin{figurecaptionlist}%
\begin{figurecaption}{pic-revcover}{on this page}% capitalised by the environment
The Night Skies Team with Chaco staff assessing the monitoring location of Gallo Cuesta\\
Photograph courtesy of Jeremy White
\end{figurecaption}

%\newline
\begin{figurecaption}{pic-cover}{on the cover}% capitalised by the environment
Fisheye view of the night sky over Chaco Culture National Historical Park in false color\\
Photograph courtesy of the National Park Service Night Skies Program
\end{figurecaption}%
\end{figurecaptionlist}%

\end{nrpsReverseCover}%
```

Figure 11: Coding for the reverse cover-page photograph and attributions. The image is linked to its caption, and vice-versa.

one sees a collection of graphic elements symbolising vegetation and wildlife, as well as scenic, recreational, historical and archaeological values [10].

When tagging is not implemented, one can simply define the expansion as follows, to just place the image normally.

```
\providecommand{\tagFigure}[3][]{#3}
```

This is best delayed using `\AtBeginDocument` as described earlier in Section 4.

For a second example where now there is some surrounding context, Figure 10 shows a photograph used inside the front cover of the research report. Notice how the alternative text appears within a popup 'tool-tip'. The image is followed, down the page, by a 'caption' and attribution of the source of the photograph, as well as similar information for the image used on the front cover. As such, we actually have a list of figure captions, rather than a single caption following each figure. Indeed these 'captions' are structurally more like 'end-notes' for the two images on either side of the cover page.

The coding to place this material is shown in Figure 11, using new commands and environments defined as follows.

```
\newenvironment{nrpsReverseCover}{%
 \newgeometry{margin=1in}\strut\vfill}{}
\newenvironment{figurecaptionlist}%
 {}{\unskip}%

\makeatletter
\newenvironment{figurecaption}[2]%
 {\noindent\footnotesize
  \Hy@raisedlink{%
   \hypertarget{#1-caption}{}}%
  \hyperlink{#1}{{\bfseries
   \MakeUppercase{#2}}\\}%
 {\unskip}
\makeatother
```

As seen in this coding, commands `\hypertarget` and `\hyperlink`, from the hyperref package, are used to link each caption to a target destination located just above the corresponding image (e.g., named `pic-revcover`), as well as linking from the image to caption (with destination `pic-revcover-caption`). This explains the detailed tagging as seen in Figure 10, which is in accordance with the recommendations in [6, §4.2.7.1], and is analogous to Example C found there. Another way to see the alternative text is upon export as text, or into XML as follows.

```
<Div>
<Link><Figure Alt="Team members silhouetted against
 night sky shortly after a vivid sunset.">

<ImageData src="images/A3-Main-pdftex_img_2.jpg"/>
</Figure>
</Link>

<L>
<LI>
<Lbl>
<Link>ON THIS PAGE</Link>
</Lbl>

<LBody>The Night Skies Team with Chaco staff
 assessing the monitoring location of Gallo Cuesta
 Photograph courtesy of Jeremy White</LBody>
</LI>

<LI>
<Lbl>
<Link>ON THE COVER</Link>
</Lbl>

<LBody>Fisheye view of the night sky over Chaco
 Culture National Historical Park in false color
 Photograph courtesy of the National Park Service
  Night Skies Program</LBody>
</LI>
</L>
</Div>
```

The above XML version gives an idea of the kind of tagged information that would be passed to Assistive Technology applications.

6 Metadata & Titlepage

One cannot downplay the rôle that Metadata plays in helping to allow documents to be found on the internet, via search engines, and otherwise located in document repositories. The more accurate and detailed the Metadata, the easier it becomes to decide whether a given document is the one that most appropriately provides the knowledge that one may be seeking. In the case of PDF documents, by Metadata we mean not only the information displayed visually on the title-page, but also keywords, copyright status and internally generated structural information, such as creation/modification dates and more.

For documents conforming to PDF/A archival specifications, the XMP packet [13] is the method that allows all such pieces of Metadata to be included in a way that can be easily extracted in XML format. It is as though the PDF document is carrying along its own Library of Congress Catalogue card. Figure 12 shows a view of part of this information, with many fields filled. Most of these fields are *not* provided automatically by LaTeX. Even more fields are shown in Figure 13, along with coding that writes information into an external file named `\jobname.xmpdata` based on the name of the LaTeX source file (in this case A3-main-pdftex.tex). This file is used by the pdfx package to create the full XMP packet, by inserting the information into a template (named pdfa.xmp), creating a new file pdfa.xmpi that is then embedded uncompressed into the PDF/A file being built using

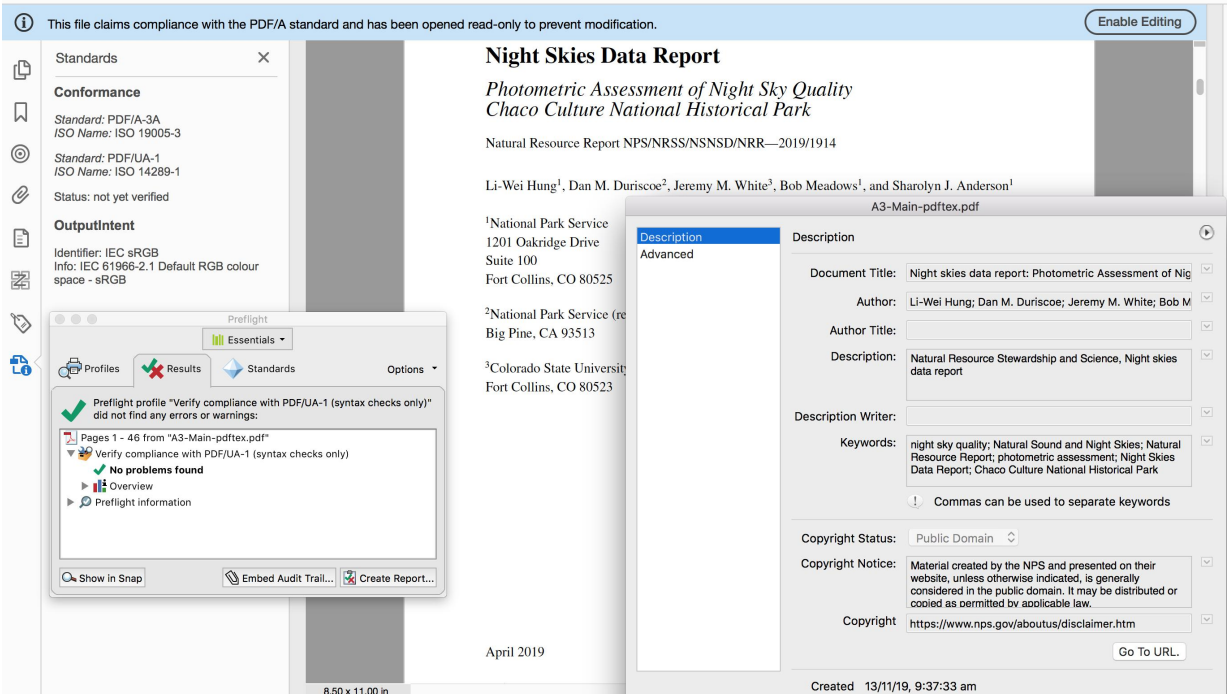
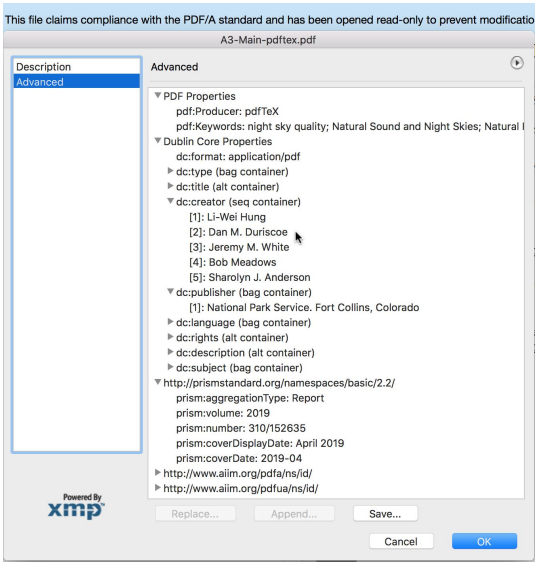


Figure 12: Metadata as shown on the title-page, as well as extra PDF `/Info` entries; conformance for PDF/A-3A and PDF/UA-1 can be checked using Preflight.




```
\providecommand{\pdfxopts}{a-3a,ua-1,pdf17,nocharset}

\begin{filecontents*}{\jobname.xmpdata}
\Title{Night skies data report: Photometric Assessment
 of Night Sky Quality \textemdash\ Chaco Culture
 National Historical Park}
\Author{Li-Wei Hung\sep Dan M. Duriscoe\sep
 Jeremy M. White\sep  Bob Meadows\sep
 Sharolyn J. Anderson}
\Publisher{National Park Service. Fort Collins, Colorado}
\Subject{Natural Resource Stewardship and Science, Night
 skies data report}
\Keywords{night sky quality\sep Natural Sound and Night
 Skies\sep Natural Resource Report\sep photometric
 assessment\sep Night Skies Data Report\sep Chaco Culture
 National Historical Park}
\Copyrighted{False}
\Copyright{Material created by the NPS and presented
 on their website, unless otherwise indicated, is
 generally considered in the public domain. It may be
 distributed or copied as permitted by applicable law.}
\WebStatement{https://www.nps.gov/aboutus/disclaimer.htm}
\PublicationType{Report}
\Journaltitle{\thejournal}
\Volume{\theyear}
\Issue{\theissue}
\CoverDate{\theyear-04}
\CoverDisplayDate{\thedate}
\Creator{pdfTeX + pdfx.sty with \pdfxopts\space option}
\Language{en-us}
\end{filecontents*}
```

Figure 13: Advanced Metadata pane, with LaTeX source that provides information for the XMP packet.

```
% Report title
\begin{nrpsInsideCover}
\nrpsTitle{Night Skies Data Report}\relax\\[-0.1in]%
\nrpsSubtitle{Photometric Assessment of Night Sky Quality \\[2pt]Chaco Culture National Historical Park}%
\nrpsSeries{Natural Resource Report NPS/NRSS/NSNSD/NRR---\hl{\theissueII}}%
\vspace{0.2in}

% Authors
\begin{nrpsAuthors}
\author{Li-Wei Hung\thanks{\nrpsmultidest
 National Park Service\\1201 Oakridge Drive\\Suite 100\\Fort Collins, CO 80525}},
\author{Dan M. Duriscoe\thanks{National Park Service (retired)\\Big Pine, CA 93513}},
\author{Jeremy M. White\thanks{Colorado State University\\Fort Collins, CO 80523}},
\author{Bob Meadows\setcounter{mpfootnote}{0}\mpfootnotemark}, and
\author{Sharolyn J. Anderson\setcounter{mpfootnote}{0}\mpfootnotemark}%
\end{nrpsAuthors}%

\nrpsDate{April 2019}
\begin{nrpsAddress}%
U.S. Department of the Interior\\
National Park Service\\
Natural Resource Stewardship and Science\\
Fort Collins, Colorado%
\end{nrpsAddress}

\end{nrpsInsideCover}
```

Figure 14: LaTeX coding for the title-page, using well-named environments as defined in Figure 16.

the pdfTeX processing engine. The package options given as `\pdfxopts` are used via

```
\usepackage[\pdfxopts]{pdfx}
```

resulting in the appropriate Metadata to declare the document to be valid for ISO standards PDF/A-3A (ISO 19005-3:2012) [3] and PDF/UA (ISO 14289-1:2012) [4], as can be seen in the left-most panel of Figure 12. These standards are built upon the PDF 1.7 (ISO 32000-1:2008) specification [14]. The claims of validation can be checked, using the Preflight utility [18] of Acrobat Pro DC [17].

It is worth remarking here that use of the pdfx package requires loading hyperref early, to be able to use some of its coding structures for writing directly into the PDF file being built. This can have consequences for the order in which other packages need to be loaded, since hyperref patches many existing LaTeX commands, to include hypertext features. One example of this is discussed and resolved, later in Section 6.1. Doubtless others will occur with other document classes, and the packages used.

Some of the Metadata that is used both on the title-page and within the XMP packet is given using macros, as discussed already in Section 4. With multiple authors and keywords, the `\sep` macro is used as a delimiter; it expands differently in various contexts. For example, the keywords ultimately occur as follows, using RDF syntax [19].

```
<dc:subject><rdf:Bag>
  <rdf:li>night sky quality</rdf:li>
  <rdf:li>Natural Sound and Night Skies</rdf:li>
  <rdf:li>Natural Resource Report</rdf:li>
  <rdf:li>photometric assessment</rdf:li>
  <rdf:li>Night Skies Data Report</rdf:li>
  <rdf:li>Chaco Culture National Historical Park
    </rdf:li></rdf:Bag></dc:subject>
```

If more, or different, Metadata fields are required for a particular class of documents, then a pdfa.xmp template can be edited appropriately. If you have a need to do this, contact the author of this article.

6.1 Authors and footnotes

In many LaTeX document classes a construction like

```
\author{ ... \thanks{ ... }}
```

provides an author name and affiliation. Normally the argument of `\thanks` is separated from the author and is typeset as a footnote. When provided in the document preamble, each use of `\author` causes the information to be stored away, awaiting use in a command like `\maketitle`. With the research report, the title-page is built separately, after the cover pages. There is no need to store the author names, but the syntax for providing that information can still be used, for consistency with authoring in other documents.

Figure 14 shows the coding for the title-page, using macros and environments named according to the principles discussed in Section 2, 3 and 4. Those names can be mapped to structure tagging as evident in Figure 15. The visual layout is determined by the macro definitions shown in Figure 16. Of note is

the nrpsAuthors environment which uses a minipage to ensure the correct style and placement of footnotes created from \thanks commands. The counter mpfootnote, used for these footnotes, avoids interference with those outside the minipage. Also within the nrpsAuthors environment we see that \author is defined to just typeset its argument, with \thanks becoming just \footnote.

Observe how the order of structural tagging need not coincide with the visual order of information on the page. Rather the order for tagging follows the logical order of occurrence in the LaTeX source, with a corresponding address following each author. A hyperlink, with anchor text being the raised footnote number, connects to each author's address. This is in accordance with [6, §4.2.7.1 Example C], but without back links as the author and address are adjacent in the tagging structure.

Where different authors share the same address (in this case authors 1, 4 and 5), the anchor text looks the same but there are separate hyperlinks to named destinations Hfootnote.1, Hfootnote.4 and Hfootnote.5. These latter two are placed together with the former, using a macro \nrpsmultidest which occurs at the beginning of the first \thanks (see Figure 14) and is defined as follows.

```
\def\nrpsmultidest{\Hy@raisedlink{%
 \hyper@@anchor{Hfootnote.4}{\relax}%
 \hyper@@anchor{Hfootnote.5}{\relax}}}%
```

Here \Hy@raisedlink and \hyper@@anchor are internally defined by the hyperref package, and are used when automatically placing hyperlink anchors. For authors 4 and 5 the \thanks is replaced by

```
\setcounter{mpfootnote}{0}\mpfootnotemark
```

to get the correct raised number, using a command \mpfootnotemark from the footmisc package. This affects just the hyperlink anchor text, as the target destination is generated using yet another counter. Loading the footmisc package is not as easy as it would seem, since it patches the LaTeX command \@footnotetext, which has been patched already by hyperref to generate the hyperlinks. Without proper care, one gets messages such as the following, for non-minipage footnotes.

```
pdfTeX warning (dest): name{Hfootnote.6}
has been referenced but does not exist,
replaced by a fixed one
```

This indicates that a hyperlink cannot work as intended. It is essentially the same issue as reported previously [20, 21]. To resolve the incompatibility we can use the ideas from Sections 2.1 and 2.2. Consider the following coding when loading the package.

```
\makeatletter
 \let\LTX@footnotetext\H@@footnotetext
 \let\HYP@footnotetext\@footnotetext
 \usepackage{footmisc}
 \let\FM@footnotetext\@footnotetext
 \let\H@@footnotetext\FM@footnotetext
 \let\@footnotetext\HYP@footnotetext
\makeatother
```

The resulting expansion for \@footnotetext is the same as if footmisc had been loaded first. It works since hyperref created a pointer \H@@footnotetext to the expansion of \@footnotetext before patching, and uses this within its own patch. What is needed is to make \H@@footnotetext point instead to footmisc's version as \FN@footnotetext, while still using hyperref's coding which has been captured as \HYP@footnotetext. This is what the sequence of three \let instances achieves, without defining any new code blocks, and we have a pointer to each defined code block. If we wish to retain pointers to only expansions that will actually be used, then this coding can be reduced by two lines. (How?)

6.2 More front-matter structures

Other pages in the front-matter section of the report have paragraphs, or other structures, that have a special semantic meaning. For example Figure 17 has a few ordinary paragraphs, but also an example citation, and one paragraph with hyperlinks to the NPS website and email address. The whole page can be wrapped in an environment nrpsPolicy that resets the page-numbering and its style, and finishes with the publication date. Although no special formatting is required for the example citation, it is worth tagging this for its semantic meaning.

```
\newenvironment{nrpsPolicy}{%
 \pagenumbering{roman}%
 \setcounter{page}{2}%
 }{\unskip \vfill \nrpsPutDate}
\newenvironment{nrpsCitation}{}{\unskip}
```

Consider again Figure 5. There is a bookmark to the 'Table of Contents' (ToC) page, but no corresponding entry within the ToC itself. Achieve this via a macro \suppresscontents which is defined within the following block of coding.

```
\makeatletter
 \def\contentssuppress{%
  \protect\contentsline{chapter}%
  {\contentsname}{\thepage}%
  {\@currentHref}\protected@file@percent}
\makeatother

\let\LTXaddtocontents\addtocontents
\newcommand{\addtocontentssuppressed}[2]{%
 \begingroup \def\thisarg{#2}%
```

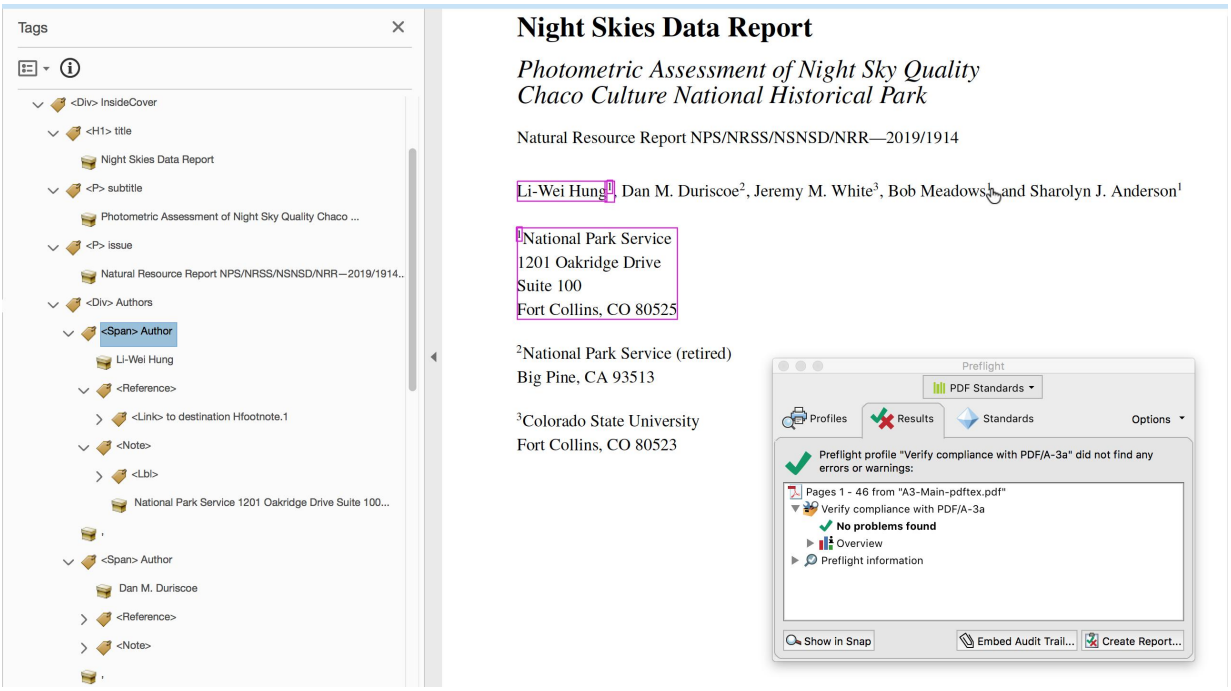


Figure 15: Tagging and layout of author information, produced using the coding shown in Figure 14. The Preflight window shows validation for PDF/A-3a.

```
\newenvironment{nrpsInsideCover}{%
 \vspace{2pt}%
 \rule{\textwidth}{1pt}\\[0.2in]}{}

% Recreations of NRR MS Word styles
\newcommand{\nrpsTitle}[1]{\noindent\fontsize{20}{24}\selectfont\textbf{#1}}
\newcommand{\nrpsSubtitle}[2][{\\[4pt]}]{\fontsize{18}{21}\selectfont\textit{#1#2}}
\newcommand{\nrpsSeries}[2][{\\[12pt]}]{\fontsize{12}{14}\selectfont{#1#2}}

% provide macro as a place-holder for structural tagging
\newcommand\nrpsAuthor[1]{#1}%

\newenvironment{nrpsAuthors}{%
% Do all the work inside a minipage,
% this allows footnotes to come immediately afterwards
% rather than at the bottom of the page.
 \begin{minipage}{\textwidth}%
  % number the footnotes with numerals
  \renewcommand{\thempfootnote}{\arabic{mpfootnote}}%
  \let\footnotesize\normalsize            % write footnotes at normal size
  \def\footnoterule{\vskip-\medskipamount}%  remove the rule above footnotes
  \footnotesep=26pt      % set spacing between footnotes
  \footnotemargin=4pt    % indent the footnote marker, else it overlaps the margin
  \parindent=0pt         % ensure no paragraph indent
  % allow \author  to be used for declaring author names
  \let\author\nrpsAuthor
  \let\thanks\footnote
 }{\end{minipage}}

% set Date and Address, with appropriate space before
\newcommand{\nrpsDate}[1]{\vfill{#1}}
\newenvironment{nrpsAddress}{\\\\}{\unskip}%
```

Figure 16: Macro and environment definitions, for the title-page as coded in Figure 14.

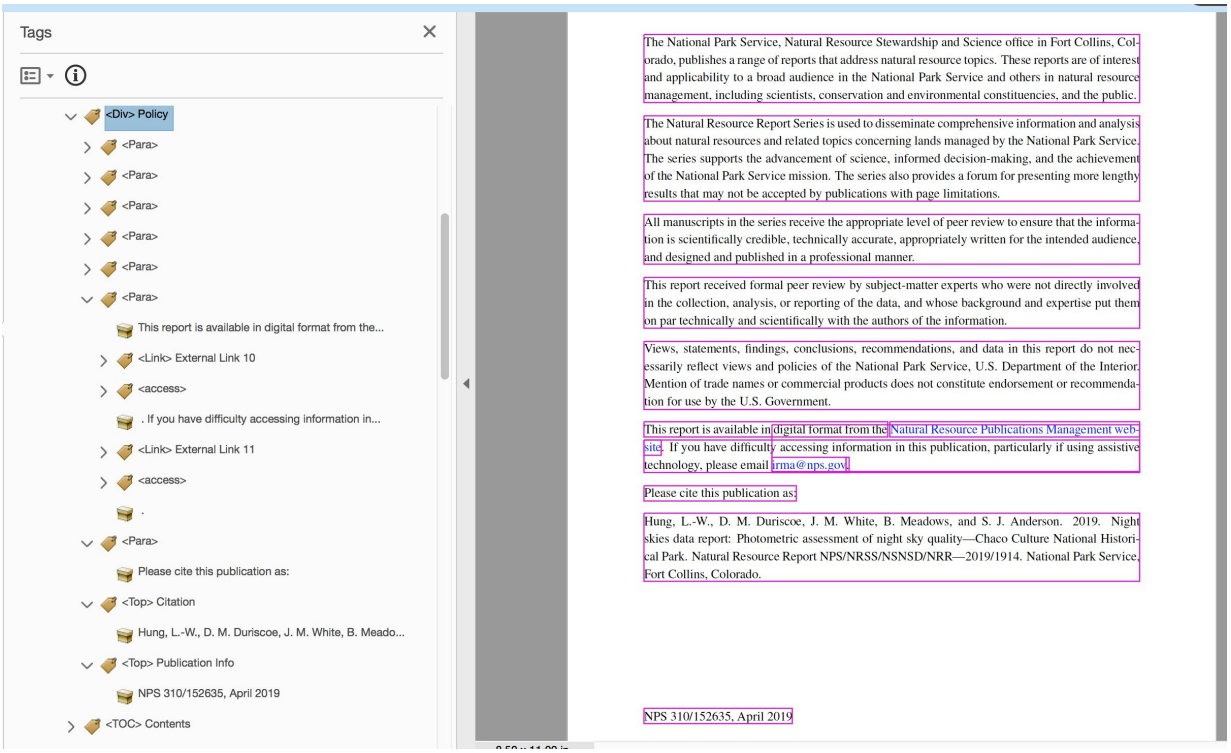


Figure 17: The page following the title-page has a paragraph with external hyperlinks, the publication date, and the recommended way to cite the report as a publication.

```
  \ifx\thisarg\contentssuppress\else
   \LTXaddtocontents{#1}{#2}%
  \fi \endgroup}%  use it locally only

\def\suppresscontents{%
 \let\addtocontents\addtocontentssuppressed}
```

The effect of this is that whenever `\addtocontents` is encountered when writing into the `.aux` file, then the argument is checked to see whether it matches what is produced by the following.

```
\addcontentsline{toc}{chapter}{\contentsname}
```

If so, then writing this ToC entry is skipped, but the `\Bookmark` command is still written into the `.out` file. On the next processing run, the desired result is produced provided the ToC is generated using coding as follows.

```
{% suppress ToC entry, but keep the bookmark
 \suppresscontents
 \maintoc
\addcontentsline{toc}{chapter}{\contentsname}
}\clearpage
```

The tpdf package patches many LaTeX macros including `\tableofcontents`, `\listoffigures` and `\listoftables`. It also patches `\@starttoc` from the tocloft package. This produces tagging in accordance with Recommendation [6, §4.1.4.1, Example B]. This includes having hyperlinks from the section titles to where the (sub-)section starts within the body of the PDF, and tagging of dot leaders as `/Artifact`, as recommended in §4.1.4.3. This can be seen, for example, in Figure 18. Page numbers are *not* hyperlinked, and the word 'Page' above the numbers is also an `/Artifact`.

Note how subsections have a nested `/TOC` structure, child of their enclosing section's `/TOCI`. This tagging is all handled automatically with no extra input required from a document's author. However there is one practical consideration of which authors and editors should be aware, when using tpdf for generating Tagged PDF documents.

The tpdf package maintains a 'tag history' file which, among other things, records the page number on which each structural and content tag occurs; or rather, on which it did occur on the completed LaTeX run. On the next run the history is read, which helps in determining internal aspects of how the tagging is structured for the pages. Also, the page number is compared with what is happening when the same tagging and content is encountered on the subsequent run. A warning message is written when there is a

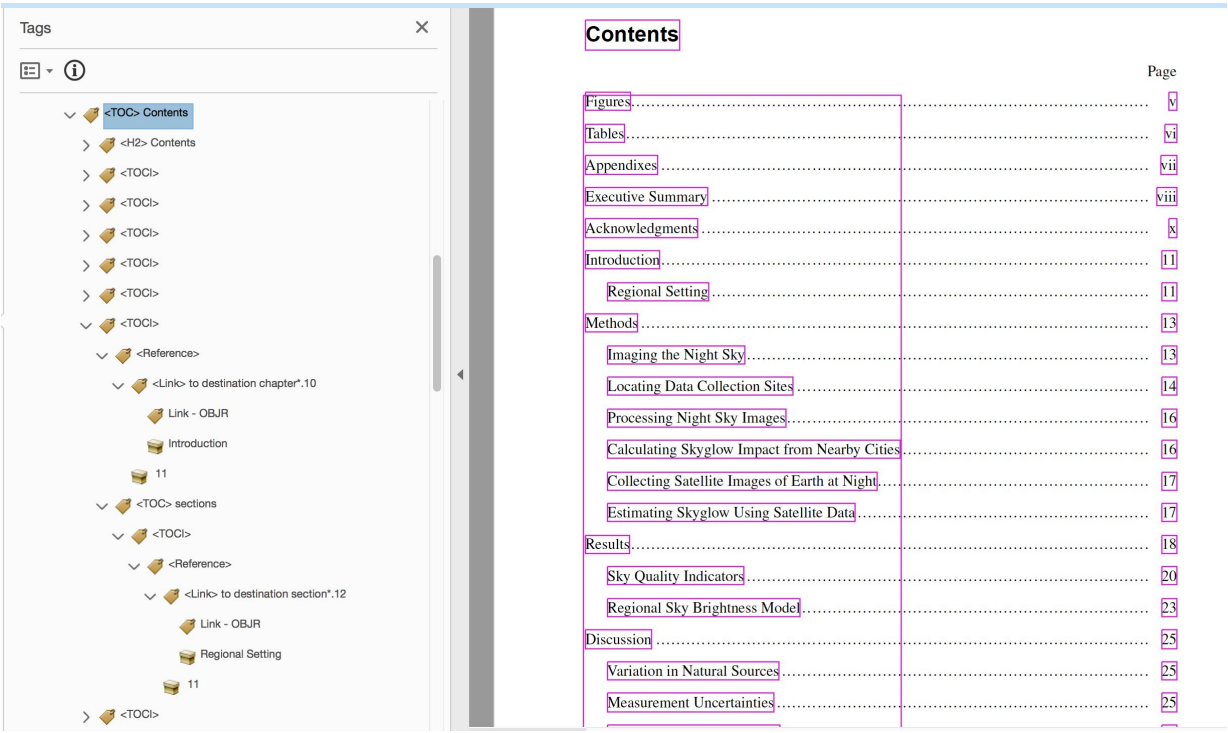


Figure 18: Fully-tagged 'Table of Contents' page, hyperlinked and with nested `/TOC` structure according to section levels.

difference. This is similar to warnings about missing or changed cross-reference or citation labels in ordinary LaTeX jobs. However the number of structural tags and content snippets is far, far greater.

As document source is being written, or edited, it is inevitable that tagging information will change, and differences in the 'history' content encountered. Thus at least one extra LaTeX run will be needed, to ensure that tagging has stabilised. In particular, as extra material is added, the length of the Table of Contents can grow, perhaps requiring an extra page. With such an extra page, there will be changes in the recorded history for all subsequent structure and content. As many as three extra runs may be required before the history has stabilised. Even then it is best to do another run 'just to be sure'. Authors and editors should get used to using the 'Console' window for interactive control of a running TeX job. As well as hitting the `return` key to continue after a pause displaying an error or warning message, one can also use the '`q`' key (followed by `return`) to switch into so-called *quiet* mode. Now there are no further messages for warnings, and the job can proceed to completion without pauses. This is most convenient on the second or third subsequent run after significant edits, when the pauses are due to detected history changes, as these will be updated without need for further edits. In case of real typos, in a macro name say, there is also '`i`' to allow insertion of the intended macro name. This is preferable to '`x`' as it can allow the job to complete without the need for multiple extra runs before stabilisation can be achieved. Just one more run, with the typo corrected, may be sufficient.

Then, when you think the document is complete or you want to check the results of significant editing, use the Preflight utility [18] to check validation (as shown in Figures 12 and 15), and examine the tagging tree within Acrobat Pro DC [17] (as in Figures 1, 3, 6, 9, 10, 15 and 17).

7 pdfLaTeX *vs.* LuaLaTeX

With reports written collaboratively by multiple authors, it is understandable that some may prefer LuaLaTeX to process the document, while another may prefer pdfLaTeX. LuaLaTeX and pdfLaTeX are built upon the LuaTeX and pdfTeX 'engines' respectively, each loading the LaTeX format. Mostly the same (at least visual) output is able to be produced, but there can be significant differences in configuration options that are needed to actually achieve this. Here we discuss such differences relevant to the 'Night Skies' research report.

7.1 Font formats

The research report [8] style is an adaptation for LaTeX following styles developed for Microsoft Word documents. As such it uses fonts 'Times' and 'Arial' in several sizes and faces. LuaTeX is able to work directly with both OpenType (`.otf`) and TrueType (`.ttf`) font formats. OpenType supports a large number of characters including accented letters for different languages, as well as mathematical symbols and much more. On the other hand pdfTeX (generally) addresses at most 256 characters in a font, and can use the TrueType (`.ttf`) font format. The winfonts package [22] provides the support needed to use the Windows' TrueType fonts with pdfTeX, but it does not supply the fonts themselves. These are presumed to be available already on a Windows (or other) system. Thus for the research report, there is coding in the 'settings' file read from the LaTeX preamble as follows.

```
\usepackage{ifluatex}
\ifluatex
 \usepackage{fontspec}
\fi

\ifluatex
%This is Overleaf Specific (or if the fonts
% are not installed in your system)
%-----------------------------------------------
% Times New Roman
\setromanfont[
BoldFont=Font-timesbd.ttf,
ItalicFont=Font-timesi.ttf,
BoldItalicFont=Font-timesbi.ttf,
]{Font-times.ttf}

% Arial
\setsansfont[
BoldFont=Font-arialbd.ttf,
ItalicFont=Font-ariali.ttf,
BoldItalicFont=Font-arialbi.ttf
]{Font-arial.ttf}

%% those TTF fonts do not validate for PDF/A :
%%  incomplete CIDSet
\else
% but all is well with pdfTeX and corresponding
%  Type-1 fonts
 \usepackage[T1]{fontenc}
 \usepackage{times}%  actually  NimbusRomNo9L
 \usepackage{winfonts}
 \renewcommand{\sfdefault}{arial}
\fi
```

This assumes that fonts named Font-times.ttf and Font-arial.ttf are available on the local system when using LuaTeX, at least via the Overleaf online system [26]. Using other systems the font file names will be different. With pdfTeX as the typesetting engine, this coding assumes packages times and winfonts are available. Packages ifluatex and fontspec come with TeX Live [25] distributions and times also, as part of LaTeX's NFSS support, whereas winfonts does not.

There is, however, a small complication with winfonts, in the form of a mistake in the virtual font for '*Arial Bold Italic*', which is used for sub-section headings. The virtual font for this actually refers to '*Arial Bold*', omitting the '*Italic*' part. A fix for this is available through a package fix-winf [23].

Alternatively, there is a package urw-arial [24] which provides Type 1 fonts based on the 'Arial' glyph shapes, provided freely by URW. But there are slight differences in glyph metrics between these and Microsoft's own TrueType fonts for 'Arial'. Furthermore, the package winfonts also loads the textcomp package, whose utility is discussed within the next sub-section.

7.2 Inline mathematics

There is one feature in LuaTeX that is different to all other TeX engines, with regard to mathematical symbols, in particular for an inline mathematical expression. A macro token like `\pm` normally only works smoothly in so-called 'math-mode', otherwise there is an error message in the Console window.

```
Missing $ inserted.
<inserted text>
                $
l.40 token like \pm
                    normally only works
?
```

Furthermore, TeX has switched into math-mode for the benefit of further mathematical symbols that may be following. The reason for this behaviour is that spacing between letters and symbols tends to be different in math-mode than with ordinary textual input into paragraphs.

On the other hand, with LuaTeX it is perfectly OK to use the '±' *character* directly in the input, as it lies within the range of characters supported by most 8-bit fonts. This behaviour, which is similar to what one might expect from other word- or text-processing software, can be very convenient where the math-expressions are short and uncomplicated. It does however bypass the very fine-tuning of spacing that otherwise results from using math-mode.

Provided the textcomp package is loaded this approach also works with pdfTeX, otherwise there is an error message as shown in Figure 19. The textcomp package sets up a mapping which produces `\IeC{\textpm}` upon reading the special character. Here `\textpm` specifies the desired character from a re-encoded (TS1) version of the current text font.

```
./testinput.tex:53: Package inputenc Error: Unicode character ± (U+00B1)
(inputenc)                not set up for use with LaTeX.

See the inputenc package documentation for explanation.
Type  H <return>  for immediate help.
 ...

l.53 pdf\TeX\ and the ±
                       character used directly on
?
```

Figure 19: Error message, using pdfTEX, resulting from input of a UTF-8 mathematical symbol, unless the textcomp package has been loaded.

Now when it comes to accessibility, it's not at all clear how a screen-reader will handle the resulting '±' character. So we add an extra layer which specifies *alternative text* for that symbol, and uses proper math-mode where appropriate. For example, the research report's preamble has coding as follows.

```
\ifluatex
\else
% account for math chars in text
 \let\LTXarcdeg\arcdeg
 \let\LTXcdot\cdot
 \let\LTXsim\sim
 \let\LTXtextpm\textpm

 \def\NRRarcdeg{\TPDFensuremath{degree}
   {\LTXarcdeg}}
 \def\NRRcdot{\TPDFensuremath{times}{\LTXcdot}}
 \def\NRRsim{\TPDFensuremath{approximately}
   {\LTXsim}}
 \def\NRRtextpm{\TPDFensuremath{plus or minus}
   {\pm}}

\AtBeginDocument{%
 \let\arcdeg\NRRarcdeg
 \let\cdot\NRRcdot
 \let\sim\NRRsim
 \let\textpm\NRRtextpm
}
\fi  %  end of  \ifluatex
```

As a default expansion, we have that

```
 \def\TPDFensuremath #1#2{\ensuremath{#2}}
```

where TEX's \ensuremath handles switching both into math-mode and back out again. With the tpdf package being used to generate Tagged PDF, the adapted command \TPDFensuremath does more by also establishing the contents of the #1 parameter as 'alternative text'. Notice how pointers have been used to capture the expansion of an existing LATEX macro, then to change the name so as to point to an enhanced code-block, as was discussed earlier in Section 2.2.

Throughout the research report such commands are used for measurements and units; *viz.*

```
...in units of W\cdot cm$^{-2}$\cdot sr$^{-1}$
...angular sensitivity is \sim 42$^\circ$.
...darker than \sim 21.5 mag/arcsec$^2$.
...brightness measurement is ±4\%.
...accurate to ±5\%, or 0.05 magnitudes.
```

Acknowledgements

Great thanks go to members of the U. S. National Parks Service science team based at Fort Collins, Colorado:

- Kurt M. Fristrup (Senior Scientist), Natural Sounds and Night Skies Division;
- Chalmers-Fagan Johnson (Web and Report Specialist);
- Li-Wei Hung (Night Skies Research Scientist), main author of the report [8];
- Damon Joyce (Night Skies Research Scientist);
- Dan M. Duriscoe, Jeremy M. White, Robert Meadows, Sharolyn J. Anderson; authors of the report [8].

for allowing use of their report as a development document for Tagged PDF using LATEX.

Thanks also to Thomas E. Price, University of Akron (emeritus), for reading an early draft and providing useful comments and suggestions.

References

[1] United States Access Board; Section 508 ICT Refresh (January 2017). §E205.4 Electronic Content. https://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-ict-refresh/final-rule/text-of-the-standards-and-guidelines#504-authoring-tools.

[2] National Science Foundation — FAQ; What is NSF's public access policy? https://www.nsf.gov/pubs/2016/nsf16009/nsf16009.jsp#q1.

[3] ISO 19005-3:2012; Document Management — Electronic document file format for long term preservation — Part 3: Use of ISO 32000-1 with support for embedded files (PDF/A-3). https://www.iso.org/standard/57229.html.

[4] ISO 14289-1:2012; Document management applications — Electronic document file format enhancement for accessibility — Part 1: Use of ISO 32000-1 (PDF/UA-1). Technical Committee ISO/TC 171/SC 2. http://www.iso.org/iso/catalogue_detail.htm?csnumber=64599.

[5] WCAG 2.1; 'Web Content Accessibility Guidelines'. W3C Recommendation 05 June 2018. W3C Consortium. https://www.w3.org/TR/WCAG21/.

[6] 'Tagged PDF Best Practice Guide: Syntax'. PDF Association. For developers implementing ISO 14289-1 (PDF/UA). Version 1.0 (June 2019). https://www.pdfa.org/resource/tagged-pdf-best-practice-guide-syntax/

[7] Moore, R.; 'LaTeX 508 — creating accessible PDFs'. Conference talk presented at TUG 2019 in Palo Alto, July 2019, delivered remotely via Zoom. Movies and other related material can be found at http://web.science.mq.edu.au/~ross/TaggedPDF/TUG2019-movies/.

[8] Hung, L.-W., D. M. Duriscoe, J. M. White, B. Meadows, and S. J. Anderson. 2019. Night skies data report: Photometric assessment of night sky quality — Chaco Culture National Historical Park. Natural Resource Report NPS/NRSS/NSNSD/NRR — 2019/1914. National Park Service, Fort Collins, Colorado. https://irma.nps.gov/DataStore/Reference/Profile/2260171.

[9] Natural Resource Stewardship and Science Directorate. National Park Service. https://www.nps.gov/orgs/1778/index.htm.

[10] 'History of the NPS Arrowhead'. National Park Service. https://www.nps.gov/glac/learn/news/history-of-the-nps-arrowhead.htm.

[11] Web AIM; Web accessibility in mind: 'Alternative Text' page. https://webaim.org/techniques/alttext/.

[12] Accessible U: 'Alt Text'. University of Minnesota web site. https://accessibility.umn.edu/core-skills/alt-text.

[13] Adobe Systems Inc.; Extensible Metadata Platform (XMP). ISO standard (16684-1). https://www.adobe.com/products/xmp.html.

[14] Adobe Systems Inc.; PDF Reference 1.7, November 2006. Also available as [15]. https://www.adobe.com/devnet/pdf/pdf_reference.html.

[15] ISO 32000-1:2008; Document management — Portable document format (PDF 1.7); Technical Committee ISO/TC 171/SC 2 (July 2008). http://www.iso.org/iso/catalogue_detail?csnumber=51502.

[16] ISO 32000-2:2017; Document management — Portable document format — Part 2: PDF 2.0 Technical Committee ISO/TC 171/SC 2 (July 2017). https://www.iso.org/standard/63534.html.

[17] Adobe Systems Inc.; Acrobat Pro DC. The complete PDF solution for any device. (Windows and Mac. only) https://acrobat.adobe.com/au/en/acrobat.html.

[18] Adobe Systems Inc.; 'Analyzing documents with the Preflight tool (Acrobat Pro)'. https://helpx.adobe.com/au/acrobat/using/analyzing-documents-preflight-tool-acrobat.html.

[19] W3C; Resource Description Framework (RDF). W3C Recommendation 25 February 2014. RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ RDF 1.1 XML Syntax. https://www.w3.org/TR/rdf-syntax-grammar/.

[20] TeX StackExchange; 'Package footmisc causes pdfTeX error'. question asked October 2015. https://tex.stackexchange.com/questions/268504/package-footmisc-causes-pdftex-error/516607.

[21] TeX StackExchange; 'Footnotes misbehaving in report go to front page but behave correctly in other report'. question asked September 2014. https://tex.stackexchange.com/questions/203439.

[22] CTAN archive: winfonts — a package to use the basic Windows TrueType fonts. Package and documentation by Paul Pichaureau, Paris, January 2006. Comprehensive TeX Archive Network. https://ctan.org/pkg/winfonts.

[23] CTAN archive: fix-winf — fixes for Windows font usage with pdfLaTeX. Package and documentation by Ross Moore, December 2019. (to appear) Comprehensive TeX Archive Network. https://ctan.org/pkg/fix-winf.

[24] CTAN archive: urw-arial — URW Arial font pack for use with LaTeX. Package and documentation by Walter Schmidt, March 2006. https://ctan.org/pkg/urw-arial.

[25] TeX Live — document production system. Distributed by TeX user groups worldwide, since 1996. Major releases annually. https://tug.org/texlive.

[26] Overleaf, Online LaTeX Editor. https://www.overleaf.com/about.

◇ Ross Moore
Macquarie University
Sydney, Australia
ross.moore (at) mq dot edu dot au

TUG 2020 abstracts

Editor's note: Videos are or will be available for most presentations; links and other information at `tug.org/tug2020`.

– – * – –

TEX and LATEX: The user experience

Jonathan Fine

Where are we? How did we get here? What's the future? I'll try to answer these questions, by looking outward in both space and time.

Don Knuth started TEX in 1977. The present version is a direct descendant of the 1982 version. Only at the end of the 1980s did hard disc drives cost less than $10 a megabyte. Today, for about $100 I can buy a pocket computer that connects to a ubiquitous network. It has gigabytes of solid state storage. It fits in my pocket, more easily than a book.

Today software that implements Don Knuth's wonderful mathematical typesetting algorithm and fonts can be downloaded for free. And this software installs itself, and is interactive. In real time it previews LATEX-encoded mathematics, as I type it.

My monitor is 40 inch, 3840x2160 with 24 bit color. It's no more expensive than a TV. (In fact, it is a TV.) It cost about $450. It's not my virtual desktop. It's my vertical desktop, about the same size as my horizontal desktop.

As often as not, when I read beautifully typeset mathematics, it's on my vertical desktop, as part of a web page. The only time I actually need a PDF is to send a file to be printed. So that I can put it on my horizontal desktop, and write on it with a pen.

And the interactive mathematical typesetting software. It's built on HTML5, and it's called MathJax. And the future of TEX and LATEX and our community. To succeed, we have to change, and also keep things the same. Linus Torvalds did something much the same with Unix, to create Linux.

By the way, my wonderful pocket computer that typesets mathematics. It's also known as a mobile phone, and it runs the Linux kernel.

Learning LATEX (and other languages) online

Jonathan Fine

Online education is suddenly more important, from primary schools to research. This talk focuses on beginners learning LATEX. We learn from what's already been done for other computer languages.

First I survey how to provide online LATEX typesetting via a web browser. They are: LATEX as a cloud service; LATEX running in the browser; MathJax running in the browser.

Next, what to teach the student, and how. Teaching is not the same as writing a reference manual. It requires identifying core concepts, and presenting them in a helpful manner and order. Students need to test their understanding, and perhaps explore, before moving on. We consider LATEX from this point of view.

At the school level, the Raspberry Pi Foundation (RPF) (`raspberrypi.org`) is one of the leaders. They offer many projects in HTML, CSS, JavaScript and Python. They have strong connections with students and teachers. They partner with Code Club (`CodeClub.org`) and Future Learn (`FutureLearn.com`). They provide many free resources. They run education research seminars. And they design and produce the Raspberry Pi.

Finally, we blend the experience of the RPF with the task of learning and teaching LATEX, and conclude with some problems, opportunities and challenges. More information: `jfine2358.github.io`.

Teaching with LATEX and Overleaf

Paul Gessler

When universities and other schools closed campuses to help reduce the spread of coronavirus, many professors and teachers quickly adapted to online teaching by necessity. Likewise, students adapted to online learning and found ways to collaborate with peers while following social distancing guidelines. Overleaf, an online writing platform for TEX, has proven helpful in many of these scenarios. This talk will provide an overview of how Overleaf can be used most effectively in an education context. Topics include: how to effectively organize projects; suggested workflows for sharing assignment templates and receiving completed assignments; using Overleaf's reviewing tools to collaborate and provide feedback on assignments.

The creative evolution of type specimens

Amelia Hugill-Fontanel

Type specimens have been produced from the earliest days of Western printing history as commercial documents designed to sell printing type. Over the course of five centuries, this publication genre has kept pace with the needs of its consumers through innovative change. This time-traveling tour will navigate through the vast holdings of the RIT Cary Graphic Arts Collection to explore the formats and features that unfurl through the history of type specimens.

Bio: Amelia Fontanel is a curator at the RIT Cary Graphic Arts Collection, a renowned library that collects on design, typography, and the book arts. As manager of the Cary technology collection, she is responsible for teaching and maintaining over 30 different presses and thousands of fonts of metal

and wood type. She is actively involved in the international printing community, holding executive board positions with the American Printing History Association and the Hamilton Wood Type and Printing Museum.

TEX and global mathematics

Patrick Ion

TEX was developed as a way of communicating mathematics. It has been very successful for that and much more. But TEX did not completely dominate publishing, though it much expanded the community able to write mathematics directly. MathML (Mathematics Markup Language) was specified as a markup for mathematics in the W3C (World Wide Web Consortium) context; it is both officially part of the web's basic HTML and an ISO standard. The idea that there should be a Global Digital Mathematics Library (GDML) is an obvious one. There's an International Mathematical Knowledge Trust (IMKT) devoted to eventually realizing a GDML, growing out of efforts by the International Mathematical Union. Some of how the present situation came to be and what's evolving now will be examined. For a historical marker see my article *MathML: A key to math on the web*, *TUGboat* 20:3 (1999), pp.167–175, `tug.org/TUGboat/tb20-3/tb64ion.pdf`.

HarfBuzz in LuaLATEX

Marcel Krüger

Starting with TEX Live 2020, LuaLATEX uses the `luahbtex` engine instead of `luatex`, and therefore allows the use of HarfBuzz instead of the ConTEXt-derived font shaper. This presentation tries to answer some of the most important questions about this change, e.g.,

- How does this affect existing documents?
- How can this system be used?
- How does this help typesetting of scripts with which LuaLATEX has always had problems?
- Why would I want to use this even for documents in scripts which are well supported by the existing shaper?
- In which cases would you not want to use HarfBuzz?
- How does this compare to XETEX's font handling?

MetaPost-based, dynamic extensible delimiters for LuaTEX

Marcel Krüger

TEX's math mode has always had support for extensible delimiters which can grow according to the content delimited by them, but these have been based on vertically stacking repeated parts. While this can often provide decent results, it leaves a lot to be desired, especially for round or angle brackets. Given that LuaTEX includes, with `luamplib`, a MetaFont-derived system, it should be possible to dynamically instantiate meta fonts with exactly the right delimiter size, without any restrictions regarding their composition. This presentation shows one implementation of this, which builds on MetaType1/AlgoType in order to output high quality, and potentially even fully hinted, vector glyphs which properly integrate into a modern LuaTEX document.

Pandoc for TEXnicians — TUG 2020 keynote address

John MacFarlane

I will give an overview of the document conversion program pandoc (`pandoc.org`), with an emphasis on how it might be useful to people who are already comfortable using LATEX to prepare documents. In the first part, I'll discuss the use of `pandoc` to convert between LATEX and other common formats, including Microsoft Word `docx` and HTML. In the second part, I'll give some reasons why even a seasoned TEXnician might want to consider writing documents in `pandoc`'s extended Markdown instead of LATEX, and I'll teach some tricks that can be used to recover the tremendous power and flexibility of LATEX in this simpler idiom.

CMaps, Virtual fonts, ActualText for reliable text extraction and accessibility

Ross Moore

The single most critical factor for document content to be accessible is that text can be extracted reliably and accurately. For a PDF file, the CMap structure gives a mapping of each character in a font to a corresponding Unicode code point. The pdfTEX and `dvipdfmx` engines have different ways to attach a CMap resource to font instances within PDF files. While it is a vital piece, the CMap is not the whole story; since the same character from the same font can be used in different ways. This is most apparent in a "fake" small-caps font, where uppercase glyphs are drawn from the same base font, but at reduced size using the virtual font mechanism. By defining a second virtual font instance, and attaching a customised CMap file and map file entry, the lowercase letters of a faked small-caps font can be correctly extracted as lowercase Latin letters.

Accented Latin letters are often constructed within virtual fonts by placing the accent first, then the base. This is counter to Unicode where the combining accent character comes after the base. By rearranging the virtual font description this order can be changed, allowing text-extraction of correctly

accented characters. This fixes many difficulties with a LaTeX T1-encoded font, but some Extended Latin characters still need further consideration. Using a little-known trick of inserting DVI special commands into the virtual font description, ActualText replacement tagging can be encoded inside the virtual font, allowing constructed characters to be mapped to their proper Unicode point.

Making a new TeX Live release available on Overleaf

Eric Mc Sween

Overleaf is an online collaborative editor for LaTeX. It produces PDF documents using a full TeX Live installation to compile projects authored by its users. Every year, when a new TeX Live version is released, it needs to be integrated in Overleaf without breaking existing projects that worked with previous TeX Live versions. This talk will explain how this is done. We will also take the opportunity to look at how the compilation service works.

TeX Live 2020 news; `texlive.info` services

Norbert Preining

TeX Live 2020 has seen the usual bunch of fixes and new version, but also one more significant change we have been working on for a long time: the renaming of containers to include the revision. The main aim of this change was to make life of distributors who rely on unique names easier. We will report a bit on the necessary changes and its implications.

The second part of the talk will briefly introduce the TeX-related services at `texlive.info`.

Authoring accessible documents, including with Ti*k*Z diagrams

Thomas Price, Ross Moore

There are two parts to this presentation. Firstly, Tom Price will describe a bundle of LaTeX files designed to build PDF/UA accessible documents from LaTeX sources using a pdfLaTeX engine. The bundle takes full advantage of the capabilities of the `pdfx.sty` and `tpdf.sty` packages while requiring minimal effort on the part of document authors.

Next, Ross Moore, author of `pdfx.sty` and `tpdf.sty`, will discuss how tagging can be achieved within a diagram created using `tikz.sty` package methods. It is becoming increasingly common to encounter images built this way, so it will be necessary to tag the information in these, so that it becomes accessible to readers with visual disabilities. Ross will demonstrate some promising first steps in this direction; in particular for a 'SWOT analysis' diagram.

Typesetting with Python

Brandon Rhodes

What would an algorithm look like that improves on the TeX typesetting system's scheme for breaking a book's text into pages? This talk explores a new Python library for high quality typesetting that I been crafting, and that I have already used to format and print a short-run hardback book.

TopTeX, a new Q&A site for TeX

samcarter

TopTeX (`TopAnswers.xyz/tex`) is a new site for questions and answers about TeX and friends. It is part of the `TopAnswers.xyz` network, an open source and not for profit project. Its development is focused on the needs of the users and provides a friendly environment for building a high quality repository of knowledge.

The newest changes to newtx and its relatives and codependents

Michael Sharpe

The `newtx` package has undergone very substantial changes over the last couple of years, while striving to remain backwardly compatible with earlier versions. In this presentation, I'll try to outline the motivations behind the changes and make comparisons with other general purpose LaTeX math packages. Among the codependency issues are problems in adapting a math package to fit a text font package, and, to a lesser extent, vice versa.

dePSFrag, the final nail in the coffin

Paulo Ney de Souza, Vadim Ponomarev

Annotation of graphics in TeX has always been a difficult subject. Solutions starting with WARMReader and passing by PSFrag and pinlabel did not have a path to more modern TeX engines (pdf-, xe- and lua-). In this talk we present a framework for moving the source files of the most common of all these packages — PSFrag — over to any other labelling desired (pinLabel, Overpic, XYOverPic and Ti*k*Z), providing a path for processing legacy content and allowing more choices in production environments.

Die TEXnische Komödie 2–3/2020

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`dante.de`). Non-technical items are omitted.

Die TEXnische Komödie 2/2020

Herbert Voss, Die Eingabe von Sonderzeichen [The entry of special characters]; pp. 6–16

The entry of special characters via the keyboard is handled differently by the various systems. As a rule, the user has no direct information about possible keyboard shortcuts, to reach certain characters, for example the German opening and closing quotes — differentiating these special characters from the character that is output by Shift+2. The latter is actually only reserved for "programming". In this article we show a list of various special keys and how they are entered on Windows, GNU/Linux and Mac OS.

Wolfgang Beinert, Schriftwahl [Font selection]; pp. 17–23

The choice of a suitable font has great importance in all sub-disciplines of typography. It not only significantly influences the readability and aesthetics of a communication medium, it also causes sustainable conclusions in the implementation. The goal of professional font selection is to find typefaces which are ideally suited for a certain task and for a certain medium without creating opportunity costs in typesetting, in production, in publication, in legal consequences, or for the recipient.

Lukas C. Bossert, Mit `biber -tool` Bibliografieeinträge bearbeiten [Edit bibliography entries with `biber -tool`]; pp. 24—32

There are several programs that can be used to edit bib files (JabRef, BibDesk, etc.). However, if one would like to create bibliography entries in an automated process, another tool is needed. This article shows how to use `biber` in tool mode to edit bib files of any size.

Herbert Voss, Eine weitere Schrift für Menschen mit Leseschwäche [Another font for people with reading difficulties]; pp. 33–35

Digital fonts for people with reading difficulties are rare. In this article a new font is introduced.

Lukas C. Bossert, Kommentieren und Dokumentieren von Code [Different ways to comment and document code]; pp. 36–48

This article is dedicated to Herbert Voß on the occasion of his farewell from the editors of the DTK. In his position at the editorial office or in seminars or on TeX.SE he has helped not only me but many others with helpful comments regarding LATEX.

Herbert Voss, Kommaseparierte Listen als Tabellen und Grafiken darstellen [Comma-separated lists represented as tables and graphics]; pp. 49–54

We show how to typeset a list of comma separated values (CSV) as a table with the help of `pgfplotstable`, and as a graphic using `pgfplots`.

Die TEXnische Komödie 3/2020

Doris Behrendt, Mario Haustein, Johannes Hielscher, Nils Pickert, Henning Hraban Ramm, CCCamp19; pp. 23–26

Report on the DANTE presence at the Chaos Communication Camp 2019 of the Chaos Computer Clubs (CCC) in the Mildenberg brick park.

Alexander Krumeich, n-doc – ein LATEX-basiertes Verfahren für IT-Sicherheitszertifizierungen [n-doc — a LATEX-based procedure for IT security certifications]; pp. 37–48

Devices and software in the environment of the telematics infrastructure of healthcare systems (TI) must meet high safety standards. In the case of the eHealth connector for connecting doctors' practices and hospitals to the TI, this requires certification according to Common Criteria (CC) by the Federal Office for Security in Information Technology (BSI).

Pascal Braband, Professionell präsentieren mit der Beamer-Klasse [Professional presentation with the `beamer` class]; pp. 48–51

LATEX in combination with the beamer class provides a powerful tool to create impressive professional (scientific) presentations. But when it comes to the actual presentation of these presentations, the possibilities are often limited. As a standard feature often only a simple PDF viewer is available, with which the slides can be displayed. This may be sufficient for simple presentations, but for more complex, longer presentations, more functions may be useful. Exactly for these requirements there are a few programs that are specialized to professionally present PDF files made with the `beamer` class. Three such solutions are presented in this article.

Herbert Voss, Chaotische Symmetrien mit Lua berechnet [Chaotic symmetries calculated with Lua]; pp. 51–57

Symmetries in chaos can be particularly well displayed graphically. Using TEXLua as a program, it does not even require the installation of the script language Lua, which is installed by default in a full installation of TEX. The graphics can be processed from a LATEX document as well as externally with TEXLua.

[Received from Herbert Voß.]

Zpravodaj 2020/1–2

Zpravodaj is the journal of $\mathcal{CS}$TUG, the TEX user group oriented mainly but not entirely to the Czech and Slovak languages. The full issue can be downloaded at `cstug.cz/bulletin`.

The issue includes several pages of photos from a visit to Don Knuth's home during TUG 2020, and Don's visit to Brno. Also, videos for the q&a sessions at Brno (item below) are linked at `tug.org/videos`.

Petr Sojka, Úvodník staronového předsedy [Introductory words from the once and future president]; pp. 1–11

This editorial introduces the content of the issue, and the author gives some personal reminiscences about Don's trips, together with reporting on the recent visit of the Grand Wizard to Brno.

Go forth and participate in $\mathcal{CS}$TUG to make the bright future of TEX & Friends a reality! You can!

Marian Genčev, Vícejazyčné pseudonáhodné generování písemných testů z databází [Multilingual pseudorandomly generated tests from databases]; pp. 12–47

The aim of this paper is the description of the class `ngt.cls` that was created to simplify the preparation of written tests for educators with common user knowledge of LATEX. The described simplification consists mainly of pseudo-random generation of tests from a prepared database of problems. Further advantages of the created system include the ease of control for the end user and the possibility of creating a version with or without results. Writing the problems in the database file is designed to work with any number of language versions in a single source file, with easy switching between them.

Vít Novotný, Markdown 2.8.1: Směle k trůnu odlehčeného značkování v TEXu [Markdown 2.8.1: Boldly unto the throne of lightweight markup in TEX]; pp. 48–56

Markdown is a lightweight markup language that makes it easy to write structurally simple documents. Existing tools for rendering markdown documents to PDF treat TEX as a black box. In contrast, the Markdown package provides support for styling and typesetting markdown documents inside TEX, extending a TEXie's toolbox rather than forcing them to replace TEX with a more limited tool.

Since its release in 2016, the package has received several important updates improving the functionality and user experience. In this article, I will reintroduce the package, and describe its new functionality and documentation.

Jan Šustek, Zpracování dat z tabulkového editoru TEXem [Processing spreadsheet data in TEX]; pp. 57–63

In the paper we show a way to read and process spreadsheet data in TEX. The macros are described in detail, allowing readers to create simple macros easily. We also show a three-line macro for inserting a whole table into a TEX document.

Tomáš Szaniszlo, Dva bloky otázok a odpovedí od Donalda Knutha na FI MU [Two questions and answer sessions by Donald Knuth at FI MU]; pp. 64–97

In October 2019 the Faculty of Informatics, Masaryk University hosted Donald Knuth as a guest who led two question and answer sessions for the occasion, dedicated to the themes of computer science and art. Following some background on these lectures, you can find their transcripts in this article.

Peter Wilson, Mělo by to fungovat IX – Opakování textu [It Might Work IX — Repetition of text]; pp. 98–104

[Printed in *TUGboat* **34**:1. Translated to Czech by Jan Šustek.]

[Received from Vít Novotný.]

Editor's note: The following questions were among those asked during the two Q&A sessions with Don Knuth (item noted above).

Session 1

- What is your favorite problem in Computer Science?
- Do you still write any code? If so, why and what computer language?
- Did you try to write a program for a quantum computer?
- What was your subject in your PhD thesis?
- What do you think about Artificial Intelligence?
- Could you tell us the story of TEX from the very beginning to implementation?
- What would you advise to your 25-year-old self?
- Is it too late for 20-year-old students to start learning real mathematics and programming?
- Biggest challenge of becoming a good programmer?
- Do you prefer screen and keyboard or paper and pencil?

Session 2

- P vs. NP (reprise)
- What's your idea about limits, the capacity of the human mind?
- Compatibility problem between musical styles in *Fantasia Apocalyptica*?

- Tabs or spaces? Vim or Emacs?
- What is the worst code you have ever seen?
- Any problems you gave up on trying to solve?
- Can you elaborate your stance on software patents?
- What would you like to redo?
- If you had the ability to write `CWEB` or `WEB` again today, would you do anything differently?
- What kind of organ do you have at home?

Photos courtesy of *Zpravodaj*;
credits to Martina Morávková and Petr Sojka.

Dragon curve (with error) in entrance hall (Don's home).

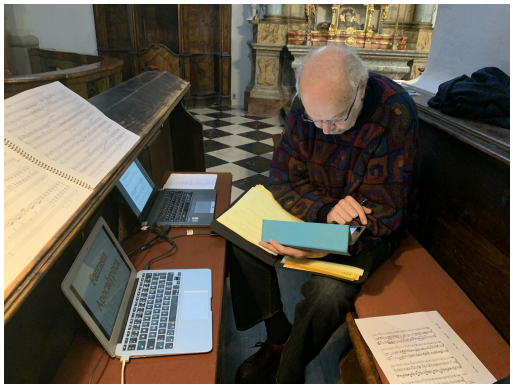
Don Knuth playing on his home organ.

P ≠ NP?, lecture of 2019-10-08, FI MU.

Type case with mementos (Don's home).

Signing after the lecture.

Don in his office.



Preparing for *Fantasia Apocalyptica* concert, 2019-11-10.

The Treasure Chest

These are the new packages posted to CTAN (`ctan.org`) from March–August 2020, along with a few notable updates. Descriptions are based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. More information about any package can be found at `ctan.org/pkg/`*pkgname*. A few entries which the editors subjectively believe to be of especially wide interest or otherwise notable are starred (*); of course, this is not intended to slight the other contributions.

We hope this column helps people access the vast amount of material available through CTAN and the distributions. See also `ctan.org/topic`. Comments are welcome, as always.

◇ Karl Berry
`tugboat (at) tug dot org`

fonts

`charissil` in `fonts`
: CharisSIL fonts with LaTeX support.

`courierten` in `fonts`
: Courier 10 Pitch BT with LaTeX support.

`ektype-tanka` in `fonts`
: Devanagari fonts from EkType.

`helmholtz-ellis-ji-notation` in `fonts`
: Inline microtonal just intonation accidentals.

`ibarra` in `fonts`
: Ibarra Real Nova fonts, with LaTeX support.

`kpfonts-otf` in `fonts`
: OpenType implementation of `kpfonts` for LuaTeX and XeTeX.

*`notomath` in `fonts`
: Math support for Noto fonts via `newtxmath`.

graphics

`bookshelf` in `fonts`
: Create a nice image from a BibTeX file.

`commutative-diagrams` in `graphics/pgf/contrib`
: Commutative diagrams using Ti*k*Z.

`dpcircling` in `graphics/pgf/contrib`
: Decorated text boxes using Ti*k*Z.

`nimsticks` in `graphics/pgf/contrib`
: Draws sticks for multi-pile Nim games.

`tikz-lake-fig` in `graphics/pgf/contrib`
: Schematic diagrams of lakes.

`tikz-planets` in `graphics/pgf/contrib`
: Illustrate celestial mechanics and the solar system.

`tikzpackets` in `graphics/pgf/contrib`
: Display network packets.

info

`dtk-bibliography` in `info`
: Bibliography for *Die TeXnische Komödie*, the journal of the German-speaking TeX user group.

`install-latex-guide-zh-cn` in `info`
: Introduction to LaTeX installation, in Chinese.

*`tex-nutshell` in `info`
: Concise document about principles of TeX.

`tlmgrbasics` in `info`
: Commonly-used actions and options for `tlmgr`.

language/japanese

`jlreq-deluxe` in `language/japanese`
: Multi-weight Japanese font support for `jlreq`.

language/marathi

`marathi` in `language/marathi`
: Marathi language support, for XeLaTeX and LuaLaTeX.

macros/generic

`expkv-cs` in `macros/generic`
: Define expandable key=val macros using `expkv`.

`expkv-opt` in `macros/generic`
: Parse class and package options using `expkv`.

`namedef` in `macros/generic`
: TeX definitions with named parameters.

macros/latex/contrib

`akshar` in `macros/latex/contrib`
: Support for syllables in Devanagari.

`algpseudocodex` in `macros/latex/contrib`
: Typeset pseudocode, based on `algorithmicx`.

`annee-scolaire` in `macros/latex/contrib`
: Typeset the French academic year.

`anonymous-acm` in `macros/latex/contrib`
: Make anonymous versions of ACM articles.

`antanilipsum` in `macros/latex/contrib`
: Italian supercazzole in Amici Miei style.

`bubblesort` in `macros/latex/contrib`
: Configurable bubble sort implementation.

`ccool` in `macros/latex/contrib`
: Encoding notational conventions.

`chhaya` in `macros/latex/contrib`
: Linguistic glossing in Marathi.

`conditext` in `macros/latex/contrib`
: Define and manage conditional content.

diabetes-logbook in macros/latex/contrib
Logbook for people with type one diabetes.
edichokey in macros/latex/contrib
Typeset dichotomous identification keys.
endnotes-hy in macros/latex/contrib
Patch endnotes for correct hyperlink anchors.
epigraph-keys in macros/latex/contrib
Epigraphs with key/value interface.
exesheet in macros/latex/contrib
Typesetting exercise or exam sheets.
frpseudocode in macros/latex/contrib
French translation of algorithmicx.
glossaries-nynorsk in macros/latex/contrib
Norwegian Nynorsk support for glossaries.
hvarabic in macros/latex/contrib
Macros for right-to-left typesetting.
ltx4yt in macros/latex/contrib
Play YouTube videos in the default browser.
media4svg in macros/latex/contrib
Multimedia inclusion for dvisvgm.
membranecomputing in macros/latex/contrib
Membrane computing notation.
menucard in macros/latex/contrib
Typesetting simple menus.
mercatormap in macros/latex/contrib
Spherical Mercator coordinate systems and Web Mercator tile integration.
metanorma in macros/latex/contrib
Metanorma standardization documents.
mlmath in macros/latex/contrib
Math notation for machine learning.
musical in macros/latex/contrib
Typeset (musical) theatre scripts.
qrbill in macros/latex/contrib
Create QR bills per Swiss payment standards.
readablecv in macros/latex/contrib
Attractive CV and letter class.
schooldocs in macros/latex/contrib
Variety of layout styles for school documents.
semantex in macros/latex/contrib
Object-oriented mathematics.
shtthesis in macros/latex/contrib
Unofficial thesis template for ShanghaiTech U.
tile-graphic in macros/latex/contrib
Generate tiles of an image.
utf8add in macros/latex/contrib
Additional support for UTF-8 LaTeX input, including math.
vcell in macros/latex/contrib
Vertical alignment of content inside table cells.
verifiche in macros/latex/contrib
Typeset Italian high school tests.
willowtreebook in macros/latex/contrib
Easy book class, built on memoir.

m/l/c/beamer-contrib/beamerappendixnote

macros/latex/contrib/beamer-contrib

beamerappendixnote in m/l/c/beamer-contrib
Create notes on appendix frames in Beamer.
beamertheme-pure-minimalistic in m/l/c/b-c/themes
Minimalist Beamer theme.
beamerthemenord in m/l/c/b-c/themes
Beamer theme using the Nord color scheme

macros/latex/contrib/biblatex-contrib

biblatex-software in m/l/c/biblatex-contrib
BibLaTeX styles for software.
biblatex-unified in m/l/c/biblatex-contrib
Unified stylesheet for linguistics journals.
biblatex-vancouver in m/l/c/biblatex-contrib
Vancouver style for BibLaTeX.

macros/luatex

lua-uni-algos in macros/luatex/generic
Unicode algorithms for LuaTeX.

macros/luatex/latex

ekdosis in macros/luatex/latex
TEI XML-compliant critical editions.
emojicite in macros/luatex/latex
Add emojis to citations.
luaprogtable in macros/luatex/latex
Programmable table interface for LuaLaTeX.
unitconv in macros/luatex/latex
Convert a length with one unit into another.

macros/plain

*zztex in macros/plain/contrib
Full-featured TeX macro package for books, journals, manuals, more.

support

git-latexdiff in support
Call latexdiff on two Git revisions of a file.
spix in support
Yet another TeX compilation tool: simple, human readable, no option, no magic.
tikztosvg in support
Shell script to render Ti*k*Z to SVG.
xml2pmx in support
Convert MusicXML to PMX and MusiXTeX.

web

pwebmac in web
Consolidated WEB macros for both DVI and PDF output.

Calendar

2020

Sep 6 – 12	14th International ConTEXt Meeting, Prague-Sibřina, Czech Republic. meeting.contextgarden.net/2020
Sep 13 – 18	XML Summer School, St Edmund Hall, Oxford University, Oxford, UK. xmlsummerschool.com
Sep 24 – 27	Ladies of Letterpress #9 is online. ladiesofletterpress.com/conference
Sep 29 – Oct 2	20th ACM Symposium on Document Engineering, to be held online. doceng.org/doceng2020
Oct 15	*TUGboat* **41**:3, submission deadline.
Oct 27 – 31	Association Typographique Internationale, ATypI All Over, www.atypi.org
Nov 5 – 8	AwayzGoose, an online gathering for lovers of type and letterpress, Hamilton Wood Type & Printing Museum *and* American Printing History Association, Two Rivers, Wisconsin. woodtype.org/pages/wayzgoose

2021

Mar 1	**TUG election:** nominations due, 07:00 a.m. PST. tug.org/election
Mar 1	*TUGboat* **42**:1, submission deadline.
Mar 10 – 12	DANTE 2021 Frühjahrstagung and 64th meeting, 32 Jahre DANTE e.V., Otto-von-Guericke Universität, Magdeburg, Germany. www.dante.de/veranstaltungen
May 2 – 5	CODEX VIII, "EXTRACTION: Art on the Edge of the Abyss", Richmond, California. www.codexfoundation.org
Jun ? – Jul ?	TypeParis21, intensive type design program, Paris, France. typeparis.com
Jun 30 – Jul 2	Nineteenth International Conference on New Directions in the Humanities, "Critical Thinking, Soft Skills, and Technology", Universidad Complutense Madrid, Spain. thehumanities.com/2021-conference
Jul ??	International Society for the History and Theory of Intellectual Property (ISHTIP), 12th Annual Workshop, "Landmarks of Intellectual Property". Bournemouth University, UK. www.ishtip.org/?p=1027
Jul 20 – 21	Centre for Printing History & Culture, CPHC/Print Networks Conference, "A visitor attraction: printing for tourists", Appleby-in-Westmorland, Cumbria, UK. www.cphc.org.uk/events
Jul 26 – 30	Digital Humanities 2021, Alliance of Digital Humanities Organizations, Tokyo, Japan. adho.org/conference
Aug 2 – 6	Balisage: The Markup Conference, Rockville, Maryland. www.balisage.net
Aug 18 – 22	TypeCon 2021, Philadelphia, Pennsylvania. typecon.com
Sep 10	The Updike Prize for Student Type Design, application deadline, 5:00 p.m. EST. Providence Public Library, Providence, Rhode Island. prov.pub/updikeprize
Oct 1 – 3	Oak Knoll Fest XXI, "Women in the Book Arts", New Castle, Delaware. www.oakknoll.com/fest

Owing to the COVID-19 pandemic, schedules may change. Check the websites for details.

Status as of 15 August 2020

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 815 301-3568, email: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

User group meeting announcements are posted at tug.org/meetings.html. Interested users can subscribe and/or post to the related mailing list, and are encouraged to do so.

Other calendars of typographic interest are linked from tug.org/calendar.html.

2021 TEX Users Group election

TUG Elections Committee

The terms of TUG President and ten other members of the Board of Directors will expire as of the 2021 Annual Meeting, expected to be held in July or August 2021.

The terms of these directors will expire in 2021: Karl Berry, Johannes Braams, Kaja Christiansen, Taco Hoekwater, Klaus Höppner, Frank Mittelbach, Ross Moore, Arthur Rosendahl, Will Robertson, Herbert Voß.

Continuing directors, with terms ending in 2023: Barbara Beeton, Jim Hefferon, Norbert Preining.

The election to choose the new President and Board members will be held in early Spring of 2021. Nominations for these openings are now invited. A nomination form is on this page; forms may also be obtained from the TUG office or via `tug.org/election`.

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

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office; the petition and all signatures must be received by the deadline stated below. A candidate's membership dues for 2021 must be paid before the nomination deadline. The term of President is two years, and the term of TUG Board member is four years.

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

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

07:00 a.m. PST, 1 March 2021

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

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

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

2021 TUG Election — Nomination Form

Eligibility requirements:

- TUG members whose dues for 2021 have been paid.
- Signatures of two (2) members in good standing at the time they sign the nomination form.
- Supplementary material to be included with the ballot: passport-size photograph, a short biography, and a statement of intent. The biography and statement together may not exceed 400 words.
- Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

Name of Nominee: _____

Signature: _____

Date: _____

for the position of (check one):

☐ **TUG President**

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

for a term beginning with the 2021 Annual Meeting.

1. _____ (please print)

   _____ (signature) _____ (date)

2. _____ (please print)

   _____ (signature) _____ (date)

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

07:00 a.m. PST, 1 March 2021.

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

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

☐ 2021 membership dues paid
☐ nomination form
☐ photograph
☐ biography/personal statement

TEX Users Group
Nominations for 2021 Election
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

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

TUG Institutional Members

TUG institutional members receive a discount on multiple memberships, site-wide electronic access, and other benefits:
`tug.org/instmem.html`
Thanks to all for their support!

Adobe Inc., *San Jose*, *California*

American Mathematical Society, *Providence*, *Rhode Island*

Association for Computing Machinery, *New York*, *New York*

Aware Software, *Newark*, *Delaware*

Center for Computing Sciences, *Bowie*, *Maryland*

CSTUG, *Praha*, *Czech Republic*

Duke University Press, *Durham*, *North Carolina*

Harris Space and Intelligence Systems, *Melbourne*, *Florida*

Hindawi Foundation, *London*, *UK*

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

Maluhy & Co., *São Paulo*, *Brazil*

Marquette University, *Milwaukee*, *Wisconsin*

Masaryk University, Faculty of Informatics, *Brno*, *Czech Republic*

Nagwa Limited, *Windsor*, *UK*

New York University, Academic Computing Facility, *New York*, *New York*

Overleaf, *London*, *UK*

StackExchange, *New York City*, *New York*

Stockholm University, Department of Mathematics, *Stockholm*, *Sweden*

TEXFolio, *Trivandrum*, *India*

Université Laval, *Ste-Foy*, *Québec*, *Canada*

University of Ontario, Institute of Technology, *Oshawa*, *Ontario*, *Canada*

University of Oslo, Institute of Informatics, *Blindern*, *Oslo*, *Norway*

VTEX UAB, *Vilnius*, *Lithuania*

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.html`. If you'd like to be listed, please see there.

Dangerous Curve
+1 213-617-8483
Email: `typesetting (at) dangerouscurve.org`
We are your macro specialists for TEX or LATEX fine typography specs beyond those of the average LATEX macro package. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical TEX and LATEX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a TEX book.

Dominici, Massimiliano
Email: `info (at) typotexnica.it`
Web: `http://www.typotexnica.it`
Our skills: layout of books, journals, articles; creation of LATEX classes and packages; graphic design; conversion between different formats of documents.

We offer our services (related to publishing in Mathematics, Physics and Humanities) for documents in Italian, English, or French. Let us know the work plan and details; we will find a customized solution. Please check our website and/or send us email for further details.

Latchman, David
2005 Eye St. Suite #6
Bakersfield, CA 93301
+1 518-951-8786
Email: `david.latchman (at) texnical-designs.com`
Web: `http://www.texnical-designs.com`
LATEX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized LATEX packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

Monsurate, Rajiv
India
Email: `tex (at) rajivmonsurate.com`
Web: `https://www.rajivmonsurate.com`
I have over two decades of experience with LATEX in STM publishing working with full-service suppliers to the major academic publishers. I've built automated typesetting and conversion systems with LATEX and rendered TEX support for a major publisher.

I offer design, typesetting and conversion services for self-publishing authors. I can help with LATEX class/package development, conversion tools and training for publishers and typesetters for book and journal production. I can also help with full-stack web development.

Sofka, Michael
8 Providence St.
Albany, NY 12203
+1 518 331-3457
Email: `michael.sofka (at) gmail.com`
Personalized, professional TEX and LATEX consulting and programming services.

I offer 30 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in TEX and LATEX: Automated document conversion; Programming in Perl, Python, C, C++, R and other languages; Writing and customizing macro packages in TEX or LATEX, `knitr`.

If you have a specialized TEX or LATEX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris
132 Warbler Ln.
Brisbane, CA 94005
+1 703 915-2406
Email: borisv (at) lk.net
Web: http://www.borisv.lk.net
TEX and LATEX consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom LATEX packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in TEX and three decades of experience in teaching & training. I have authored more than forty packages on CTAN as well as Perl packages on CPAN and R packages on CRAN, published papers in TEX-related journals, and conducted several workshops on TEX and related subjects. Among my customers have been Google, US Treasury, FAO UN, Israel Journal of Mathematics, Annals of Mathematics, Res Philosophica, Philosophers' Imprint, No Starch Press, US Army Corps of Engineers, ACM, and many others.

We recently expanded our staff and operations to provide copy-editing, cleaning and troubleshooting of TEX manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

Warde, Jake
Forest Knolls, CA, 94933, USA
+1 650-468-1393
Email: jwarde (at) wardepub.com
Web: http://myprojectnotebook.com
I have been in academic publishing for 30+ years. I was a Linguistics major at Stanford in the mid-1970s, then started a publishing career. I knew about TEX from Computer Science editors at Addison-Wesley who were using it to publish products. Beautiful, I loved the look. Not until I had immersed myself in the production side of academic publishing did I understand the contribution TEX brings to the reader experience.

Long story short, I started using TEX for exploratory projects (see the website referenced) and want to contribute to the community. Having spent a career evaluating manuscripts from many perspectives, I am here to help anyone who seeks feedback on their package documentation. It's a start while I expand my TEX skills.

Reports and notices

TUGBOAT Volume 41 (2020), No. 2

Reports and notices