TUGboat

Volume 40, Number 3 / 2019

TEX Users Group

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

Individual memberships

2019 dues for individual members are as follows:

- Trial rate for new members: $20.
- 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 organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate for 2019 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: October 2019]

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

In 1986 Knuth published *[The] Metafontbook*,
his text on typeface design. The dedication reads:
"To Hermann Zapf: Whose strokes are the best."

Jerry Kelly
Hermann Zapf and the World
He Designed (2019)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
Editor Barbara Beeton

Volume 40, Number 3, 2019
Portland, Oregon, U.S.A.

TUGboat editorial information
This regular issue (Vol. 40, No. 3) is the last issue of the 2019 volume year. The deadline for the first issue in Vol. 41 is March 31, 2020.

TUGboat is distributed as a benefit of membership to all current TUG members. It is also available to non-members in printed form through the TUG store (`tug.org/store`), and online at the *TUGboat* web site (`tug.org/TUGboat`). Online publication to non-members is delayed for one issue, to give members the benefit of early access.

Submissions to *TUGboat* are reviewed by volunteers and checked by the Editor before publication. However, the authors are assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

TUGboat advertising
For advertising rates and information, including consultant listings, contact the TUG office, or see:
`tug.org/TUGboat/advertising.html`
`tug.org/consultants.html`

Submitting items for publication
Proposals and requests for *TUGboat* articles are gratefully received. Please submit contributions by electronic mail to `TUGboat@tug.org`.

The *TUGboat* style files, for use with `plain` TEX and LATEX, are available from CTAN and the *TUGboat* web site, and are included in common TEX distributions. We also accept submissions using ConTEXt. Deadlines, templates, tips for authors, and more, is available at `tug.org/TUGboat`.

Effective with the 2005 volume year, submission of a new manuscript implies permission to publish the article, if accepted, on the *TUGboat* web site, as well as in print. Thus, the physical address you provide in the manuscript will also be available online. If you have any reservations about posting online, please notify the editors at the time of submission and we will be happy to make suitable arrangements.

Other TUG publications
TUG is interested in considering additional manuscripts for publication, such as manuals, instructional materials, documentation, or works on any other topic that might be useful to the TEX community in general.

If you have such items or know of any that you would like considered for publication, please contact the Publications Committee at `tug-pub@tug.org`.

TUG Institutional Members

TUG institutional members receive a discount on multiple memberships, site-wide electronic access, and other benefits:
`tug.org/instmem.html`
Thanks to all for their support!

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*, *Floida*

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*

TNQ, *Chennai*, *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*

From the president

Boris Veytsman

As a member of The Book Club of California (https://www.bccbooks.org), I was invited to organize an exhibition for the club members. I decided to devote it to the history of TEX.

It feels strange to talk about a history of a computer program: the art of programming still seems new and shiny. Karel Čapek, a great Czech writer, noted in 1925,

> Car drivers are always young. Some day in the future we could read in a story, "The devoted old driver grabbed the steering wheel with trembling hands", or "Old Petr, still strong despite his age, put on his best scarf to drive the bride and groom to the church". Some day they will write about "spacious heirloom cars" as they used to write about ancient carriages, and about wise old car drivers as they now sometimes write about wise old cabbies.

We are now living in the age when there are not only old (spacious!) cars, but also old computer programs (and, sadly, old programmers). It is even more strange that we are working with one of these heirloom programs, which is still working for us, daily churning out thousands of beautiful books and articles. However, after four decades, TEX is still strong (despite its age, as Karel Čapek would slyly note).

In the preparation of the exhibition I have been helped by the recent two-part series by Barbara Beeton, Karl Berry, and David Walden in the *IEEE Annals of the History of Computing* (the preprints are available at http://walden-family.com/ieee/texhistory.html).

I am also being helped by the generous TUG members who donated many items and helped me with valuable suggestions. I am especially grateful to Martin Ruckert for the source code for his program, William Adams who sent me (physically!) a large box of books, and to Dave Walden, who shared with me many rarities, including the beautiful monograph by David R. Siegel on Euler project at Stanford.

Most of all, I was absolutely stunned by the generosity of DEK, who gave me the TEX incunabula: the first ever book typeset in TEX in 1978. The book is described in Don's paper (TEX Incunabula, *TUGboat* 5:1, 1984, pp. 4–11, https://tug.org/TUGboat/tb05-1/tb09knut.pdf):

> I like to think that the first *real* book to be printed with TEX was a 28-page keepsake that was made for my wife's relatives at Christmastime, 1978. This book included eighteen original linoleum block illustrations, into which we pasted XGP-produced text set in a special 14-point extended variant of the prototype Computer Modern font. In order to compensate for the XGP's limited resolution, we prepared magnified copy and the printer reduced it to 70%; the effective resolution was therefore about 286 pixels/inch. [...] About 100 copies were printed, of which roughly 25 were sold and the remaining 75 were given as gifts. A complete library citation for this book would read as follows: "*Lena Bernice: Her Christmas in Wood County, 1895.* By Elizabeth Ann James, with illustrations by Jill Carter Knuth. Columbus, Ohio: Rainshine Press, 1978."

It is astonishing to think of the number of TUG members born after *Lena Bernice* was printed.

Typography is a conservative art. As has been noted a number of times, we call typefaces that appeared in the eighteenth century "Modern". TEX, which was first used for a "real" book in 1978, is still considered sometimes an upstart by bibliophiles — a mirror image of TEX as an old geezer among some computer people. My opinion is that it is neither: TEX is a great tool created within the long tradition of typography. The latter was always influenced by technology changes, even since movable type (a technological innovation!) was born. What makes TEX different — and very useful — is the idea that the beauty of a printed page can be understood as a function of the layout, and the function can be calculated and optimized by a computer algorithm. Since this idea transcends the details of technology available in the 1970s, TEX has turned out to be quite resilient. It has survived the introduction of PostScript, the advent of PDF, several changes of font technology and many other innovations. I am sure it will survive the current challenges and technological changes.

The thought I would like to convey in the Book Club of California exhibition is that TEX is a natural tool for a typographer, which, unlike many other technological tools, is aimed at increasing beauty rather than sacrificing it to other goals like economy or speed. *The TEXbook* starts with the description of TEX as *a new typesetting system intended for the creation of beautiful books [...]*, and its main part ends with the famous exhortation "Go forth now and create *masterpieces of the publishing art!*". I hope to be able to communicate this thought during the exhibition.

◇ Boris Veytsman
https://tug.org/TUGboat/Pres

Editorial comments

Barbara Beeton

TUG 2019 sponsors

In an effort to rush the proceedings issue to the printer, the recognition of sponsors was overlooked. The omitted information appears in this issue on pages 280–282. We are most appreciative of their support, and apologize for the omission.

Kerning between lowercase+uppercase

In the Computer Modern fonts, no kerning is defined between any lowercase letter and an adjacent capital. This combination is rare (probably nonexistent) in ordinary English words, but may occur in proper names and its frequency is growing in trademarked names and CamelCase programming notation.

A question on `tex.stackexchange` noted that the combination "fF" (the "femto-farads" unit of capacitance) seems to be typeset strangely or incorrectly.[1] Indeed, this almost appears to be a ligature, which is certainly incorrect.

The unfortunate crunching together of the two members of such a combination can be remedied manually in several ways in math mode (e.g., inside `\mathrm` in LaTeX):

- inserting an italic correction: `f\/F` → fF;
- inserting a zero kern: `f\kern0pt F` → fF;
- inserting an empty group: `f{}F` → fF.

In math mode, they all result in TeX inserting the italic correction value for 'f' (which is nonzero in the `cmr` fonts). In text mode, it is necessary to explicitly insert the italic correction with `\/`.

The last of these, empty braces, can disappear if the combination is in a string that is a moving argument (e.g., an index term) when saved and set later in a secondary location.

Differential "d"

The question "What's the proper way to typeset a differential operator?" persists. Is it an upright "d" (as specified by ISO 80000-2:2009 and its predecessor ISO 31-11 (1992)), or italic, as it appears in a couple centuries worth of math journals and books?

The question was already raised in *TUGboat*, in the 1997 article "Typesetting mathematics for science and technology according to ISO 31/XI"[2] by Claudio Beccari. That ISO standard mandates the upright form. But that standard was devised by engineers and physicists; no mathematicians were involved.

A question in the History of Science and Mathematics thread of StackExchange[3] has brought the matter up again. The supporting detail is rather extensive, and has caused me to search further. So, since I'm still investigating, this note is not the final word; look for a dedicated article in the next issue.

Bibliographic archives in BibTeX form

Although in some ways BibTeX has been overshadowed by BibLaTeX, a monumental body of bibliographic information in BibTeX form exists in curated and normalized databases and archives.

The BibNet Project archive at the University of Utah[4] is managed by Nelson Beebe. This archive continues to grow, and the sibling TeX Users Group bibliography archive now holds BibTeX data for nearly 900 journals and about 90 subject-specific bibliographies. (This includes a complete `.bib` file for all issues of *TUGboat*.) The BibNet Project archive holds additional subject-specific material and extensive bibliographies of important scientists in many fields of computing and numerical analysis.

In addition to the bibliographic data, many tools for managing such data are available as well, with extensive documentation[5] describing the BibTeX program, tools, and best practices for writing BibTeX entries.

The entire content of this archive — both data and tools — is freely available.

Nelson has presented talks at TUG meetings and written articles on the tools (and many other TeX topics) for *TUGboat*; see the cumulative *TUGboat* contents online.[6]

Other freely available bibliographies for computer science are located at the University of Karlsruhe[7] and the University of Trier.[8]

The MathSciNet[9] service of the American Mathematical Society holds bibliographic information and reviews for about 4 million books and articles dating from the 1800s, and is available to users at institutional subscribers to the service.

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

[1] https://tex.stackexchange.com/q/505607

[2] *TUGboat* **18**(1), pp. 39–48, https://tug.org/TUGboat/tb18-1/tb54becc.pdf

[3] dy/dx versus $\mathrm{d}y/\mathrm{d}x$, https://hsm.stackexchange.com/q/6727

[4] http://www.math.utah.edu/pub/bibnet
http://www.math.utah.edu/pub/tex/bib

[5] http://www.math.utah.edu/pub/bibnet/bibtex-info.html

[6] https://tug.org/TUGboat/Contents/listauthor.html#Beebe,Nelson

[7] https://liinwww.ira.uka.de/bibliography/

[8] https://dblp.uni-trier.de/

[9] https://mathscinet.ams.org/mathscinet/index.html

Ukraine at BachoTEX 2019: Thoughts and impressions

Yevhen Strakhov

In May 2019, four students of Odessa I.I. Mechnikov National University and myself, Deputy Dean of the Faculty of mathematics, physics and information technology, attended BachoTEX, the 27th annual GUST conference of TEX users and friends.

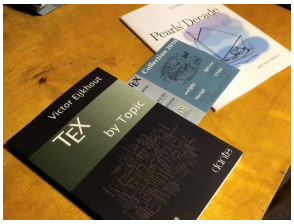
Our team at BachoTEX. Photo by Harald Koenig

First, I would like to thank the TEX Users Group, GUST and all the people who made this trip possible. We are happy to be the first Ukrainian participants at BachoTEX and look forward to further cooperation. Now I want to look back and note some important points and impressions.

Education. I have been teaching LATEX courses to students since 2014. So, first of all, we considered this trip as a good opportunity to widen our knowledge about TEX, its history and 'cutting edge'. The world is getting smaller now, and we have access to many teaching materials from the best universities and other institutions. However, despite the rapid development of online education, I think nothing can replace a real lecture and live communication. BachoTEX was a great place for communication because all lecturers were open-minded and friendly. So attending the talks brought us even more than we expected.

"This was the first time I attended the BachoTEX conference. I use TEX software for writing term papers and diplomas. At the conference, I learned that TEX's capabilities are not limited by these actions. Presentations, magazines, typing musical notes, writing the code... For me, this conference was very informative," said one of the students, Yuliia.

Popularization of TEX. In Ukraine, particularly at my alma mater, Odessa National University, we have a few TEX enthusiasts rather than an organized community. I am a big fan of TEX and try to popularize it among the students of our faculty. I use it for typesetting my books, lecture notes and other teaching materials, examination sheets and even posters, flyers and certificates. It's nice to see that more and more students have recently started using LATEX to prepare bachelor, master or PhD thesis, draw pictures with TikZ and don't fear using non-WYSIWYG systems anymore.



Events such as the BachoTEX conference clearly help the TEX community to grow and, on the other hand, also contribute to promoting TEX products among educators and students. We received many printed materials (many thanks for that) and the latest version of TEX software on DVD, which I will be able to distribute at the faculty to popularize their use. Furthermore, we met with many users and professionals of ConTEXt — I think only a few people in Ukraine have heard about this software (I will try and tell my colleagues).

International experience. Three out of four of our participants were abroad for the first time of their life. And this is no wonder. Unfortunately, due to economic and social reasons, many young Ukrainians don't have enough opportunities to visit other countries in Europe and in the world. That is why higher education institutions in Ukraine now are trying to make more international contacts to allow students to participate in exchange programs, scientific schools, workshops, etc., funded by foreign organizations. Many of today's school graduates are searching for educational programs with more international opportunities. This is the reason why participation in BachoTEX can also make a positive impact at our faculty and university.

Time for a coffee break. Photo by Ye. Strakhov

Networking. The basic idea of any conference is communication, making new friends and meeting like-minded people in your area of interest. Again, BachoTEX can bring even more, because TEX users are not only mathematicians or computer scientists — they are very versatile and creative people (with a good sense of humor)! And — for me the most important — all of them love what they do. This makes the atmosphere warm and unforgettable.

Yuliia adds, *"There were a lot of different people at the conference, so I thought it would be difficult for me to see eye to eye with everyone. They were of different ages and nationalities, they had different interests. But absolutely everyone was positive and was inspired by the conference. It was their openness and friendliness that allowed me immediately join the team."*

"We did not limit ourselves to discussing only TEX. There were many interesting contests and games, master classes, beer tasting. Enthralling tradition — Bonfire and spending the last night in house number 13."

Workshop on logo design. Photo by Ye. Strakhov

People. I cannot forget the famous quote: *"Education is what survives when what has been learned has been forgotten."* No matter who said that, there is a rationale to it. Education is not only about theoretical or practical knowledge, but also about design of thinking, emotions and impressions. It is not only about classroom activities, but also about informal events like those at BachoTEX. Education is for people and it is powered by people. And maybe the best feedback to you as an organizer and lecturer is when your students are happy.

Another member of our team, Inna, recently wrote (the following is my translation):

"To say that I am delighted with this country is not to say anything. Poland won me over with its architecture, style, cleanliness and comfort. Poles? These are people for the soul, people next to whom you just feel indescribable peace. BachoTEX? I am incredibly grateful to the people who motivated, sponsored and just participated in this event. I received a lot of positive and only positive emotions that are just unforgettable. #BachoTeX_2020 #you_my_small_dream"

It motivates the most. We, all the Ukrainian guests, would like to express our deep gratitude for the organizers' work. We highly appreciate your concern for us. Thanks to you, our stay at the conference was comfortable and we were happy. The high level of organization has greatly impressed us.

Other members seem to agree with me, *"The information content of the conference and the cordial attitude of its participants left me with the most positive opinion."*

"I think that attending BachoTEX will be a good experience for students and they will definitely get a lot of impressions."

Thanks for an opportunity to be a part of the worldwide TEX community. We hope to be here again next year.

◇ Yevhen Strakhov
Dvoryanskaya str., 2
Odessa, 65082
Ukraine
strakhov.e.m (at) onu dot edu
dot ua

An experience of trying to submit a paper in LaTeX in an XML-first world

David Walden

The publishing world is increasingly using an XML-first work flow wherein the source manuscript is first converted to XML, editing is likely done in XML, and output formats are created from what comes out of XML.[1] This note describes some recent author experience with a publisher that was moving in that direction.

The IEEE Computer Society is one of almost 40 IEEE societies. For decades, it had a strong in-house editing/publishing organization in southern California producing 13 peer-reviewed magazines and more than 30 transactions for various research areas. The Computer Society has been under financial pressure for a number of years as, like other professional societies, the Society (and the IEEE more generally) lost members. As one way of dealing with financial issues, since 2015 the Computer Society has been squeezing its editing/publishing organization through staff layoffs, out-sourcing, and so on. Simultaneously, the Society has felt the need to publish HTML, Epub, etc., versions of its magazines in addition to PDF versions and has been pushing toward non-print publication. In our current digital era, societies also have the expense of maintaining digital archives of the journal issues (e.g., `computer.org/csdl/magazine/an`). For 2018 the Society moved to largely highly hands-off editing and prepress operation. Authors were expected to use a Word template[2] to lay out their own papers for print publication which would have minimal staff copy editing.

1 Our article, part 1

In the second half of 2017 Barbara Beeton, Karl Berry, and I researched and wrote a paper on the history of TeX and its community for submission to the *IEEE Annals of the History of Computing*, a magazine of the IEEE Computer Society. The paper was to be part of a pair of special issues on desktop publishing. At the request of the special issue guest editors, we split our 17,000 word draft into two parts so one part could go in each of the pair of special issues.

Early in 2018 we began the process of submission for peer review, revision, and publication. The 2018 rules for submission for peer review required that papers be submitted in the Word template format. There was also a hint that the Society was working on a path to be able to handle LaTeX submissions.

We had drafted our paper's parts in LaTeX, and we begged to be able to submit a pair of PDFs out of LaTeX for peer review and not to have to use the Word template unless the paper was accepted for publication. We were given permission to submit the PDFs. But before submission, deciding I had to learn the new Word template anyway for other *Annals* work, I converted the paper's two parts to Word using the Society's `.dotx` Word template.[3] The template style, as shown in the example,[3] was single column ragged right that I assume was to simplify layout.

The paper was tentatively accepted, pending something between a minor and major revision. One of the reviewers strongly made the point that a paper on (La)TeX should be published looking like TeX rather than word processor output. We revised our LaTeX source that I had converted to Word for peer review, and we then volunteered to be a test case for the Society's effort to develop a LaTeX class that matched the format of the Word template.

We were told to go ahead with being a LaTeX test case, but there was no promise the class would be done in time for publication. We iterated several times with the class developer, first converting to his draft class, and then changing things in our LaTeX file as he refined the class; we also helped by testing our paper against his backend use of Pandoc to go to HTML, Epub, etc. We were then allowed to submit our paper's parts in LaTeX for copy editing. We polished parts 1 and 2 of the paper using the new class, sent them to the copy editor, and part 1 was published in *Annals* issue 2018-3.[4] Part 2 was to be published in 2019.

2 Our article, part 2

However, by the fourth quarter of 2018, more of the Computer Society editing and publications staff had been let go, and the IEEE publications staff in New Jersey took over the editing and prepress work of the Society's magazines. (The Computer Society's transactions journals had been transferred to the IEEE a few years earlier.)

We understood (incorrectly, it turned out) that a new IEEE Word template was being created for the *Annals* going forward in 2019 and work was beginning modifying an existing IEEE LaTeX class to match the style of the Word template being developed. We understood correctly that the IEEE editors accepted LaTeX input.

We converted our part 2 file from the 2018 LaTeX class to the `IEEEtran` class[5] (`ctan.org/pkg/ieeetran`) which we thought we were told would be close to the final 2019 *Annals* LaTeX class. We passed this to the IEEE editors, and we volunteered once again to be a test case as the developing *Annals*

LaTeX class was refined, but we received no response to our offer.

Then we received a proof for part 2 of our paper which was, unfortunately, unpublishable, especially for a paper on the history of TeX and LaTeX.[6] It didn't look like LaTeX output, and we couldn't elicit an explanation of how the proof had been created from our LaTeX submission. Their response was "mark up the proof and we will work with that". A next proof came, which was better but still unsatisfactory from our point of view. By this time, we had been told that they accepted LaTeX but converted it to XML and all editing after that was of the XML. We gave up on a couple of things their system seemed unable to handle, such as the TeX family of logos, and we lowered our expectations for good line breaking and inter-word spacing. A third proof was acceptable enough. The paper was published in the second 2019 issue of the *Annals*.[7,8]

3 Summary

We were unlucky that the two special *Annals* issues on desktop publishing, a year apart, spanned the time when there was a big discontinuity in the journal's staffing and style. On the other hand, this is the sort of thing that can happen with publishers trying to deal with the pressures of today's world.

If you use LaTeX because you like its beautiful output: be aware that that is not what many publishers are working on; they are working on a common path to all the desired output formats, some of which are inherently not pretty. Try to ascertain if the journal to which you submit will process your LaTeX source with LaTeX. If the journal has an XML-first publication workflow, moderate your expectations for the result.[9]

Personally, I will continue to compose papers in LaTeX because I enjoy using it (and my WinEdt editor) more than I enjoy using Word. It will be convenient if a journal to which I submit accepts the LaTeX input even if it will not be processed by LaTeX. If necessary I can convert from LaTeX to Word for submission.

4 Notes on the workflow IEEE uses for *Annals*

As I understand things, the IEEE Computer Society editors used Word — the submission format of most authors — to lay out papers for print. I believe they output to PDFs for print publication and somehow also moved from Word to HTML and Epub.

As noted above, we learned that the IEEE editors have everything converted to XML and from there papers go to print, HTML, and Epub. I was told the conversion is done by outside vendors, and the conversion methods are proprietary to the vendors. The editors also do not give out the vendors' names.[10]

References and notes

[1] Jonathan McGlone, Preserving and Publishing Digital Content Using XML Workflows, `tinyurl.com/xml-mcglone`

[2] `walden-family.com/texland/tex-xml/e1.pdf`

[3] `walden-family.com/texland/tex-xml/e7.pdf`

[4] `walden-family.com/texland/tex-xml/e6.pdf`

[5] `walden-family.com/texland/tex-xml/e3.pdf`

[6] `walden-family.com/texland/tex-xml/e4.pdf`

[7] `walden-family.com/texland/tex-xml/e5.pdf`

[8] The two parts of our peer-reviewed accepted TeX-history paper before publication are at `walden-family.com/ieee/texhistory.html`. If you would like a copy of the two parts as published, please ask me: `dave.walden.family@gmail.com`

[9] Stefan Moser, in the context of writing for the *IEEE Transactions on Information Theory*, has some guidance on submitting LaTeX which will be converted to XML: Stefan M. Moser, Author Information: How to Avoid Common Conversion Problems LaTeX — XML, August 10, 2016, `ece.umd.edu/trans-it/TIT-FinalSubmissions.pdf`

[10] Also, in the publishing process we saw, it appears that the outside vendors are not using methods with state-of-the-art line breaking and hyphenation. Also the process makes a paper's author feel distant from the copy editor(s) as they don't communicate directly.

◇ David Walden
`walden-family.com/texland`

Studying the histories of computerizing publishing and desktop publishing, 2017–2019

David Walden

The 2019 TUG conference was to be my third presentation about the history of publishing, printing, and typesetting. For TUG 2012 in Boston (where I live), I thought an appropriate topic was Printing & Publishing in Boston: An Historical Sketch.[1] This kindled an interest in digital typography history; thus, my topic in 2016 in Toronto was An Informal Look into the History of Digital Typography.[2] I then hoped to spend some time expanding my 2016 paper into a small monograph, but first an opportunity came for me to learn much more about the history of digital typography, which I decided would be my subject for TUG 2019.

In the end, I was unable to attend TUG 2019 to present this material. The slides that I did not present at TUG 2019, that go along with the content of this paper, are viewable at `tug.org/l/walden-tug19-slides`.

1 Desktop publishing meeting

I was invited to be an observer at a two-day May 2017 meeting at the Computer History Museum of pioneers of desktop publishing (DTP).[3] Pioneers in attendance at the meeting were:

Chuck Bigelow (Bigelow & Holmes type design studio)
Paul Brainerd (Aldus)
Liz Bond Crews (Xerox PARC, Adobe)
Charles Geschke (PARC, Adobe)
Steve Kirsch (Frame Technology)
Don Knuth (Stanford, TEX)
Butler Lampson (PARC)
Lee Lorenzen (Ventura Software)
John Scull (Apple)
Jonathan Seybold (Rocappi, Seybold Publications)
John Shoch (PARC)
Charles Simonyi (PARC, Microsoft)[4]
Bob Sproull (PARC)
Larry Tesler (PARC and Apple)
John Warnock (PARC, Adobe)
Richard Ying (Atex)

The meeting was organized by Burt Grad, co-founder of the museum's Software History Special Interest Group, and David Brock, director of the museum's Center for Software History. They sought advice from Jonathan Seybold about which pioneers to invite to the meeting. This was the fourteenth pioneer meeting Burt has organized or co-organized since 2003, and these meetings have resulted in eight special issues of the *IEEE Annals of the History of Computing*, additional stand-alone *IEEE Annals* articles, and 130 oral histories for the museum's collection. (Burt invited me to the meeting because he knew of my work for many years as a member of the editorial board of the *Annals*.)

This desktop publishing pioneer meeting is resulting in two (closer to three) desktop publishing special issues of the *Annals*:

Issue 1 (*Annals* vol. 40, no. 3, July–September 2018)
Desktop Publishing: Laying the Foundation
by Burton Grad and David Hemmendinger
Rocappi: Computerizing the Publishing Industry
by Jonathan W. Seybold
How Atex Helped an Industry Change the World
by Douglas Drane
More about Atex by Jonathan Seybold
The Xerox Alto Publishing Platform by Robert F. Sproull
How Modeless Editing Came To Be by Lawrence G. Tesler
The Origins of PostScript by John E. Warnock
TEX: A Branch of Desktop Publishing, Part 1
by Barbara Beeton, Karl Berry, and David Walden
Interview with Charles Bigelow by David Walden

Issue 2 (*Annals* vol. 41, no. 3, July–September 2019)
Desktop Publishing: Building the Industry
by Burton Grad and David Hemmendinger
Seybold Publications and Seminars by Jonathan Seybold
Founding and Growing Adobe Systems Inc.
by John Warnock and Charles Geschke
Paul Brainerd, Aldus Corporation and the Desktop Publishing Revolution by Suzanne Crocker
Desktop Publishing: The Killer App That Saved the Macintosh
by John Scull and Hansen Hsu
Interview with Tim Gill (Quark) by Jay Nelson
Frame Technology and FrameMaker by David J. Murray
The Ventura Story by Lee Lorenzen

TEX: A Branch of Desktop Publishing, Part 2
by Barbara Beeton, Karl Berry, and David Walden
(published in *Annals* vol. 41, no. 2, April–June 2019)
Oral History of Liz Bond Crews by Paul McJones
(to be published in *Annals* in early 2020)
Font Wars parts 1 and 2 by Charles Bigelow
(to be published in *Annals* in early 2020)

Burt Grad and David Hemmendinger are the special issue guest editors. The Computer History Museum has posted on its website the transcripts of the nine sessions of the two-day meeting. The Museum also has or soon will have interviews or oral histories of the following: Charles Bigelow, Paul Brainerd, Charles Geschke and John Warnock, Steve Kirsch, Donald Knuth, Butler Lampson, Lee Lorenzen, John Scull, Jonathan Seybold, Robert Sproull, Gary Starkweather, Larry Tesler, and Charles Thacker.

From the meeting and from reading the above materials and helping prepare them for publication, I learned about the following topics that were missing from my 2016 paper.

- Computerizing newspaper, periodical, and book publishing
  - John Seybold and Rocappi
  - Michael Barnett's PAGE-1

  - Bringing "all digital" to newspapers, e.g., Atex
- Jonathan Seybold and the Seybold Reports and Seminars
- Development of the desktop publishing technology and market: Xerox PARC, Adobe, Aldus, Apple, Frame, Interleaf, Quark, Ventura
- The "Font wars" of 1989 to 1995 and prior technology

I will sketch a bit about all but the last of these in this paper. My monograph will cover more.

As I see things, use of digital computers in typesetting and publishing followed two more or less parallel paths in the 1960s and 1970s. (1) Various commercial vendors were working to computerize the publishing industry, initially via computer control of phototypesetting machines; I discuss this in the next section. (2) Various individuals in universities and research laboratories were bringing out a succession of computer programs to format text for their typewriter printers and line printers; I discussed this in my 2016 presentation. Then in the 1980s with laser printers, page description languages such as PostScript, and "desktop publishing", the two worlds came together; I discuss this in section 3.

Table 1: Phototypesetting-to-digital for industry *

1961–64 Michael Barnett's experiments at MIT ***
1962 • John Duncan began research on computer typesetting at the University of Newcastle-upon-Tyne
• RCA 301 and IBM 1620 based hyphenation and justification at newspapers
1963–1970 John Seybold's Rocappi company ***
1964–65 IBM 1401 and 1130 and DEC PDP-8 based typesetting systems
1964 Saltzer's RUNOFF at MIT — interactive text formatting ***
1966–67 PAGE-1 computer composition system, produced in RCA's Graphics Systems Division ***
1967 ff. other similar systems
1971 Seybold Reports started by John and Jonathan Seybold ***
1973–1981 Atex offers full office newspaper/ periodical/etc. system ***

* This table is derived from a September 2018 note by Jonathan Seybold titled "Early steps in computer typesetting in the 1960s" and posted at `history.computer.org/annals/dtp/rocappi-typesetting.pdf`. Items marked *** are discussed further in the text.

2 Making publishing digital

Digital technology started coming to the typesetting world in the 1930s, when Linotype machines began to be able to input paper tapes, either coming from wire services (TeleTypeSetting) or punched on local TTS keyboards. Fax-like systems were also in use to transmit images. In the 1950s, phototypesetting systems began to be available and by the 1960s their use was spreading widely, replacing Linotype machines. The early phototypesetters were driven by paper tapes punched from a keyboard; next the typesetters were connected directly to keyboard using dedicated electronics, and then general purpose computers drove the typesetters; whole pages could then be specified — text plus layout.[5,6] Eventually the computer systems capabilities expanded to encompass all the functions involved in producing a newspaper or periodical. Throughout this process there was lots of backward compatibility. For instance, a computer system capable of complex typesetting and page layout would still need to be able to handle the electronic equivalent of paper tape input from wire services.

2.1 Going from phototypesetting to digital

There were a number of key people, newspapers, and vendors that pioneered and spread the increasingly digital technology, originally with phototypesetters but heading toward all digital. Marcus and Trimble[7] noted some of these companies, such as the *Minneapolis Star Tribune*, the *New York Daily News*, and the Mergenthaler Linotype Company.

Another useful summary of the early activities in computer typesetting comes from a note by Jonathan Seybold.[8] Table 1, derived from Seybold's note, shows some of the steps in computerizing the printing and publishing industry (newspapers, periodicals, books) in the phototypesetter era. An exception in the table is Saltzer's 1964–65 development of RUNOFF. RUNOFF's purpose was to nicely print people's documents on their personal terminals or office line printers; it was not aimed at the challenges of the publishing industry. In the rest of this section, I will say something about the efforts marked with *** in the table.

2.2 Michael Barnett

Jonathan Seybold believes[8] that Michael Barnett's document shown in Figure 1 was one of the first two documents phototypeset from output generated by a computer; the other was a press release also produced by Barnett. Wikipedia reports that Barnett also typeset a number of books with his computer composition system.[9]

Barnett was at MIT and was working with a Photon 560 "film setting" machine. Text and in-

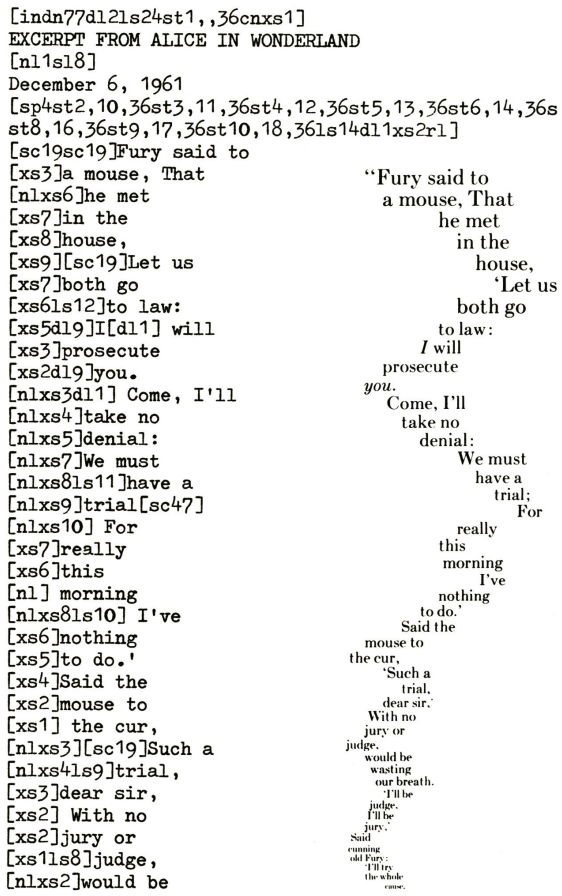


Figure 1: Barnett's reproduction of a page from chapter 3 of *Alice in Wonderland* with phototypesetter commands

structions (as shown in Figure 1) were typed on a Friden Flexowriter that output the typed characters on paper tape. This paper tape was converted by Barnett's TYPRINT program running on MIT's IBM 709 computer into another paper tape in a format understandable by the Photon 560. Another program in the 709, TABPRINT, could input paper tapes from non-Flexowriter sources.

Barnett wrote a book on his computer composition work at MIT which is widely cited.[10] His work during this 1961–64 period also apparently was useful in terms of helping other people see what they could do themselves. Barnett's book is also a useful reference for what happened before his work and suggests the state-of-the-art when he was working.

Having slipped into the domain of computer composition from physics because of a need to format some physics formulas, Barnett became further involved with the worlds of computer composition (see section 2.5), publishing, and libraries.[9]

2.3 Rocappi

John Seybold and his son Jonathan had impact on the printing and publishing industry from 1963 to 1990. They were involved with the Rocappi company from 1963 to 1970, as I discuss below. After Rocappi they (first both and then Jonathan alone) produced the Seybold reports and seminars (section 3.2). What I sketch below is primarily based on Jonathan's paper in the first *Annals* special issue on desktop publishing.[11]

John Seybold joined the fledgling world of computerized phototypesetting in 1963 with the formation of his company Rocappi (sometimes written ROCAPPI) — Research on Computer Applications in the Printing and Publishing Industries. In 1962 John knew the publishing industry well, but not computers, when he saw an early computer typesetting system at a newspaper. He immediately envisioned many ways a computer could speed the move away from hot metal type that was already underway with phototypesetting. He started his company to participate in and help advance what he saw as a coming revolution. Rocappi had inordinate impact for its size.

Rocappi didn't do research itself. Rather, it took on a variety of typesetting jobs, primarily from publishers, and used a computer to carry out the jobs under the principle that new programming for any job should be done in a general enough way that it could lead to a general class of jobs. Their software ran on an RCA 301 computer, and their software could generate instructions for various different phototypesetters (their own and their customers'). In Jonathan Seybold's paper on Rocappi,[11] he describes several aspects of the software created at Rocappi.[12]

Device independent markup. Since the different phototypesetters required different commands to drive them, a person keyboarding the text to be typeset could give a command to specify which phototypesetter the output was for (e.g., βa for the first kind of typesetter, βb for the second kind of typesetter) and then ignore phototypesetter differences in specifying text formatting commands. Also, the formatting commands were generally abstract rather than actual device codes (e.g., $hb for second level heading, $hc for third level heading), with implementations in code differing according to the document style and which typesetter was specified with the prior β command. The $-codes could also indicate an actual device code.

Pattern based hyphenation. Jonathan Seybold has sketched[11,13] the hyphenation method used by Rocappi.

> The Rocappi routine... looked at successive blocks of five consecutive letters. Each five-letter combination pointed to a position in a table of bits. If it was permissible to place a hyphen between the second and third letters, the bit would be one. If it was not permissible, the bit would be zero.
>
> The bit table was generated by running a heuristic program against a large dictionary. The program was left running overnight, night after night, until it stopped improving itself.

According to Seybold, this approach to hyphenation was used by Rocappi from its earliest days (ca. 1963) and was developed by Colin Barber ("an exceptional programmer").[14] I could not learn enough about Rocappi's method of hyphenation to compare it with Frank Liang's approach which is used in TEX.

Hyphenation correction. The Rocappi computer did not have the capacity to hold the entire dictionary. Thus they used the pattern based hyphenation method described in the prior paragraph. However, they felt that the market required perfect hyphenation. After running their H&J program, the hyphenated words in a document were sorted into alphabetical order and then compared against the dictionary to catch any mis-hyphenated words.

To deal with the book publishing market (more demanding than the newspaper market), Rocappi's typesetting system also supported kerning, tracking, and ligatures.

Character width changing. Jonathan Seybold also has described the following technique Rocappi used (nearly 30 years before Zapf and Karow's similar ideas were published):[13]

> I found at Rocappi that allowing the composition program to vary the set width of the characters on a line of type in very fine increments gave the composition program a great deal of added flexibility in producing beautifully justified type. Changing the type set width by a tenth of a point or so results in changes to character and word shapes that are imperceptible to the human eye, but which make a considerable difference over the length of a line.

Seybold is speaking of a Harris Intertype Fototronic CRT typesetter which had been modified to allow type to be sized to 1/10 of a point (1/720 of an inch).[15] Rocappi used this technique, for instance, to typeset the King James Bible.

Pagination and vertical justification. Again to cope with having a small computer, Rocappi's system scanned over a hyphenated and justified text file and extracted just enough information about the text to calculate where page breaks would go — the text itself was not needed. Using the resulting "text facts" file, the page makeup program could calculate the best places for page breaks, including calculating "vertical justification" adjustments within pages and making adjustments to prior pages that improved later pages. The program "produced a compact page descriptor file that specified what was to go on every page and what spacing adjustments were required to make the page come out right."[11]

For the Bible project, Jonathan discovered that slight changes in interline spacing within a column to make columns the same length were not noticeable to readers even if one column had one more line than the other column.

Over the life of Rocappi, the Seybolds had many connections throughout the publishing industry. Visitors came to see what they were doing. Jonathan wrote a book describing and comparing all of the then extant CRT typesetters. They did some consulting. Jonathan has written,[11] "[We] viewed Rocappi as an opening chapter in what we expected would be a revolutionary change in publishing technology. Sharing what we were trying to do was a way to help kindle that revolution."

In 1967, Rocappi was sold to Lehigh Press, with the Seybolds continuing to worki for the company. In 1970 Jonathan left Rocappi "having played a role in the embryonic states," and wanting to find a way to play a role in the bigger field that was going to move very fast. His father left the same year. (The Seybolds' story is continued in section 3.2.)

2.4 Jerome Saltzer's RUNOFF

As stated above, RUNOFF was aimed at interactive use by authors drafting and formatting their own documents rather than for use by professional typesetters in the publishing industry, which is what Barnett and Rocappi were working on. Saltzer released RUNOFF in 1964 for use under the CTSS operating system on MIT's IBM 709 and 7090 computers; it was written in the MAD programming language.

RUNOFF had only 16 formatting commands, as shown on the left in Table 2 on page 9 at `tug.org/tug2016/walden-digital.pdf`. From that list of commands and the RUNOFF output shown in Figure 4 on page 10 at the same url, you can understand RUNOFF's limited but still useful capability.

RUNOFF was influential, leading to similar programs for other computers: Script for CP/CMS, and roff, which led in turn to nroff, troff, ditroff, and groff. Over time the text formatting systems for individual interactive use that followed RUNOFF gained

in capability and became able to produce quality that would be acceptable to the publishing industry. With desktop publishing's rise in the 1980s, the two worlds came together.

2.5 PAGE-1

During 1965–66 Michael Barnett was employed by the Graphic Systems Division of RCA to develop of the PAGE-1 computer composition system.[16] The PAGE-1 system, written in assembly language for the RCA Spectra 70 computer, was released in 1967 for use with an RCA VideoComp 70|820 Electronic Photocomposer. PAGE-1 appears to have been primarily aimed at typesetting books or book-like documents. PAGE-1 was programmable in a rudimentary way (unlike RUNOFF or, I believe, Barnett's experimental work at MIT). PAGE-1's capabilities included:

- Thirteen typographic variables such as maximum interword space (**mx**), top boundary (**tb**), and the typeface in use (**tf**).
- Three read-only variables for horizontal position (**cx**), vertical position (**cy**), and current character (**cc**, decimal code for most recently set character).
- Several types of global variables:
  - page number (**pn**)
  - footnote counter (**fn**)
  - standard paragraph indentation (**pi**)
  - up to 201 general variables (**gv*n*** where $0 < n \leq 200$); almost 150 of them conventionally held particular information, e.g., point size for footnote text (**gv14**) and space between primary text and footnotes in points (**gv172**).
  - up to 9 indirect variables (**iv*i*** where $1 \leq i \leq 9$) [I don't know what these were for.]
- Six arithmetic operators; the example given for sum is as follows:
  [**ad**,variable,parameter,parameter]
- Six conditional procedures where two parameters are compared and if the condition is true, the following action is taken, for example (the operators for less-than, equal, and greater-than):
  [**lt**,parameter,parameter[text-to-set]]
  [**eq**,parameter,parameter[[code-to-be-executed]]]
  [**gt**,parameter,parameter[[code-to-be-executed]text-to-set]]
- Eight instructions within the system for editing the text; these were dropped from PAGE-2 as it was simpler to use a separate text editor to edit the source text and insert the PAGE-1 markup.
- Names for synonyms and formats:
  — Synonyms had two-character names (e.g., **x1**) where the first character is a letter in the range from t–z and the second character is a number from 1–9; these names are given to strings of text and/or code for use within a job.
  — Formats had two-character names (e.g., **a3**) where the first character is a letter in the range from a–s and the second character is a number from 1–9; these names are given to strings that are in a central library for use from job to job.
- An instruction for assigning a sequence of instructions and/or text strings to a name, for example (from the first line of Figure 2):
  [**sy,x1**[[**gt,cx,gv1**[[**gv1,cx**]];**nl**]]]
  This sequence defines **x1** as a **sy**nonym for the rest of the characters between the open and final close square brackets. Assuming **gv1** (the saved horizontal position) has previously been initialized to 0, the rest of the string does the following: If **cx** is greater than (**gt**) **gv1**, then **gv1** takes the new value of **cx**. Either way, a newline (**nl**) finishes off the line.

```
[sy,x1[[gt,cx,gv1[[gv1,cx]];nl]]]
[sy,x2[[x1;df,gv1,rb,gv1;qo,gv1,gv1,2;us]]]
...
[gv1,0;su;lb,gv1]
Some text[x1]
Some more text[x1]
And this[x2]
```

Figure 2: A small example of PAGE-1.

Figure 2 shows a bit of PAGE-1 programming (paraphrased from the Pierson book).[16] I think the example works as follows.

> The first line of this example was already explained in the description above of how the synonym instruction worked.
>
> The second line defines another synonym, x2. When the x2 code is executed, first the x1 code is executed. Then the difference (df) between rb (right boundary) and gv1 is taken and becomes the new value of gv1. Next the quotient of gv1 over 2 is taken and becomes the new value of gv1. Then, typesetting is unsuspended (us).
>
> The next instruction shown in the example first sets gv1 to zero. Then typesetting is suspended (su). Then the left boundary is set to gv1, i.e., also set to zero.
>
> After the first line of text is scanned, x1 is executed. This checks if the current character position (cx) is bigger than the current value of gv1. Since it is, i.e., cx is 9 and gv1 is zero, gv1 is set to the value of cx (9), and processing goes on to the next line.
>
> Which works the same as the prior line. At the end of the line, cx is 14 which is greater than 9, the previous value of gv1, so gv1 is set to 14, and processing goes on to the next line.

```
B3 ≡ [gv143,gv123;gv149,gv123;gv168,gv88;gv169,gv88;
      gv195,0;gv196,0;gv197,0;gv198,0;gv200,0;iv1,98;
      gv1,0;gv3,3;ad,pn,pn,1;fn,1;
      sb,6;lb,0;tb,0;rb,gv58;bb,gv88;pd,gv88;
      ne,gv123,0[[bb,gv87;pd,gv87]];ib;
      sb,2;lb,0;rb,gv58;bb,gv88;ps,gv15;bl,ps;tf,gv45;Y4;nl;
      sb,3;lb,0;rb,gv59;bl,gv29;ju;wb[[Y3]];cb;
      ab[[jv;sb,4;bb,gv99;eq,gv182,0[[ib;sb,5;cb]];gv182,0;
          iv1,99;gv1,1;gv3,5;ab[[jv;pg;sb,1;bb,gv168;ad,pn,pn,1;
          pd,gv88;ne,gv123,0[[pd,gv186;by,2,0]];ib;sb,2;lb,0;rb,gv58;
          Y2;nl;sb,3;lb,0;rb,gv59;ju;cb;by,1,gv143;by,4,gv149;
          gv98,gv168;gv99,gv169;gv144,gv143;gv150,gv149;
          gv195,gv197;gv196,gv198;gv168,gv88;gv169,gv88;
          gv143,gv123;gv149,gv123;gv197,0;gv198,0;
          gv200,0;iv1,98;gv1,0;gv3,3]]]];wb[[Y1]]]

B4 ≡ [ad,gv144,cy,gv110;gv150,gv144;nl;by,1,gv144;by,4,gv150;
      df,gv98,gv88,gv144;ne,gv123,0[[df,gv98,gv87,gv144]];
      gv99,gv98;sb,1;lb,0;rb,gv59;bb,gv98;ps,gv11;bl,gv26;tf,gv41;ju]

B6 ≡ [gv192,cy;bb,gv88;gv182,1;eq,gv1,0[[sb,5]]

B7 ≡ [eq,gv1,0[[jd;sb,4;bb,gv99;eq,gv182,0[[ib;sb,5;cb]];
              gv182,0;iv1,99;gv1,2;gv3,5]];
      eq,gv1,1[[jd;pg;sb,1;bb,gv168;ad,pn,pn,1;ib;
              sb,2;lb,0;rb,gv58;Y2;nl;sb,3;lb,0;rb,gv59;
              ju;cb;by,1,gv143;by,4,gv149;gv98,gv168;
              gv99,gv169;gv144,gv143;gv150,gv149;gv195,gv197;
              gv196,gv198;gv168,gv88;gv169,gv88;gv143,gv123;
              gv149,gv123;gv197,0;gv198,0;gv200,0;iv1,98;gv3.3]];
      df,gv1,gv1,1]

[sy,Y1[[B7]];sy,Y2[[**]];sy,Y3[[B6]];sy,Y4[[‡‡]]]
** Desired running head. †† Desired drop folio.

B8 ≡ [eq,gv200,1[[eq,gv198,0[[eq,gv197,1[[ge,gv169,gv6[[ge,gv6,gv92[[gv198,1]];
                                                lt,gv6,gv92[[ne,gv149,gv123[[gv198,1]];eq,gv149,gv123[[ad,gv149,gv149,gv6]]]];
                                                df,gv169,gv169,gv6]]];
                                    eq,gv197,0[[lt,gv168,gv6[[ge,gv169,gv6[[ge,gv6,gv92[[gv198,1]];lt,gv6,gv92[[ne,gv149,gv123[[gv198,1]];
                                                                                                                eq,gv149,gv123[[ad,gv149,gv149,gv6]]]];
                                                                                df,gv169,gv169,gv6]]];
                                                ge,gv168,gv6[[ge,gv6,gv92[[gv197,1]];
                                                lt,gv6,gv92[[ne,gv143,gv123[[gv197,1]];eq,gv143,gv123[[ad,gv143,gv143,gv6]]]];
                                                df,gv168,gv168,gv6]]]]]]];
  eq,gv200,0[[eq,gv196,1[[ge,gv6,gv92[[gv197,1]];lt,gv6,gv92[[ad,gv143,gv143,gv6]];df,gv168,gv168,gv6;gv200,1]];
            eq,gv196,0[[eq,cy,0[[gv90,0;ne,cx,0[[gv90,ps]]]];
                                    ne,cy,0[[eq,cx,0[[df,gv90,cy,bl;ad,gv90,gv90,ps]];ne,cx,0[[ad,gv90,cy,ps]]]];df,gv90,bb,gv90;
                                    eq,gv1,0[[eq,gv195,1[[ge,gv6,gv92[[gv196,1]];
                                                lt,gv6,gv92[[ne,gv150,gv123[[gv196,1]];eq,gv150,gv123[[ad,gv150,gv150,gv6;by,4,gv150]]]];
                                                df,gv99,gv99,gv6]];
                                            eq,gv195,0[[ge,gv90,gv6[[ge,gv6,gv92[[gv195,1]];
                                                                lt,gv6,gv92[[ne,gv144,gv123[[gv195,1]];
                                                                        eq,gv144,gv123[[ad,gv144,gv144,gv6;by,1,gv144]]]];
                                                                df,bb,bb,gv6;gv98,bb]];
                                                        lt,gv90,gv6[[ge,gv6,gv92[[gv196,1]];lt,gv6,gv92[[ad,gv150,gv150,gv6;by,4,gv150]];
                                                        df,gv99,gv99,gv6]]]]];
                                    eq,gv1,1[[ge,gv90,gv6[[ge,gv6,gv92[[gv196,1]];lt,gv6,gv92[[ne,gv150,gv123[[gv196,1]];
                                                                                        eq,gv150,gv123[[ad,gv150,gv150,gv6;by,4,gv150]]]];
                                                df,bb,bb,gv6;gv99,bb]];
```

lt,gv90,gv6[[ge,gv6,gv92[[gv197,1]];lt,gv6,gv92[[ad,gv143,gv143,gv(

THIS SET OF FORMAT STATEMENTS will generate two-column pages with footnotes and allocate white space for subsequent "strip in" of graphics. The format statements B3 and B4 are used together to provide for a full, two-column chapter opening followed by two columns of text to complete the chapter opening page. The format statements B7 and B6 are the text block and footnote block bottom boundary procedures respectively and they must be equated with the synonym names Y1 and Y3 in the job specification.

White space is cited by assigning to gv6 a value equal to the depth of white space desired (in points) and then calling B8; the amount of white space requested should equal an integral number of text lines.

The footnote format statements appear and their use is described on page B11.

The two-column style generator (for use with these format statements) provides for user selection of two basic typographic formats. See discussion on page B11.

Figure 3: Example PAGE-1 definitions used in marking up text with two columns, footnotes, and allocated white space for subsequent "strip in" of graphics.

The next line is processed, and x1 within the definition of x2 is executed which finds that 8 is not greater than 14, so gv1 remains 14.

Executing the rest of x2, let's suppose that the right boundary (rb) is 60. Subtracting 14 (gv1) from rb gives 46 which becomes the new value of gv1. The quotient (qo) of gv1 and 2 is 23 which becomes the new value of gv1.

Typesetting is then unsuspended (us) and apparently this causes typesetting to jump back to where it was suspended (su), i.e., in the middle of an instruction to where the left boundary (lb) is set — now to 23, the value of gv1. This second time through the three lines of text, that same calculation happens but none of it matters as the three lines all begin at position 23, calculated such that the longest bit of text is centered on a line. (I didn't try to figure out from the book how PAGE-1 deals with proportional fonts.)

Programming in TEX macros doesn't seem too hard compared to PAGE-1. A more realistic PAGE-1 example is shown in Figure 3, a page from Pierson's book.[16] The typeset text at upper right refers to the definitions on the rest of the page and elsewhere.

In time, an expanded version of PAGE-1 was developed, known as PAGE-2. Still later, Information International Inc. delivered PAGE-1 and PAGE-2 on one or two computers other than the Spectra 70. Barbara Beeton has told me that the AMS used

PAGE-2 for administrative (non-math) publications, before the Science Typographers, Inc. system which came before TEX.

2.6 Atex

In Table 1, the Seybold Reports, started in 1971, come before Atex which was started in 1973. However, even through the Seybolds are mentioned in this section, I will discuss the Seybold Reports together with the Seybold Seminars in section 3.

Jonathan Seybold has summarized nicely the desire for automation by newspapers in the early 1970s.[17]

> By the early 1970s, many reasonably sized businesses were using computers for support functions, especially for accounting, billing, inventory, and so forth. In addition to these functions, newspapers were also using small computers (IBM 1130s and DEC PDP-8s) programmed to perform hyphenation and justification (H&J) to increase productivity in the composing (typesetting) room.
>
> Next, newspapers set out to do something far more ambitious: computerize the entire process of creating and producing their product. The news copy for the newspaper would be written, edited, formatted, and composed on interactive terminals. All of the "copy flow" between writers and editors would take place within the computer system. All classified ads would be taken, priced, and composed on the same system. Ultimately, all display advertising and all page makeup would be done using interactive graphic display terminals.

Figure 4 shows a computerized newsroom.

In a 1991 report of the National Academy of Engineering, Wilson Locke gives a detailed description of the 1970s effort of the *Los Angeles Times* to computerize and the reasoning behind the effort.[18]

Living in the Boston area since 1964, I was aware of the existence of Atex, but I knew nothing about it until the May 2017 Computer History Museum desktop publishing pioneers meeting and the writings about Atex[17] that appeared in the first desktop publishing special issue of the *IEEE Annals of the History of Computing*.

Douglas Drane and brothers Charles and Richard Ying founded Atex in 1973. They met Jonathan Seybold at a national computer conference, where Jonathan learned what they intended to do. John Seybold, Jonathan's father, was consulting to *U.S. News and World Report* magazine at the time, and knew that *U.S. News* was seeking a new all-digital system such as the one the Atex partners were planning.

Figure 4: Atex terminals in *Newsday* newsroom on Long Island, 1977 [photo by John Seybold, courtesy of Jonathan Seybold].

U.S. News gambled on Atex, providing upfront funding and the specification of the system they wanted. Atex got the initial system working on *U.S. News*' tight deadline and over the next few years supplied systems to many other companies and institutions, becoming the most popular supplier of computer systems for newspapers and periodicals.

Each customer installation was a custom system based on Atex's highly efficient and relatively inexpensive hardware and software configuration, including considerable hardware they developed themselves. The Atex systems could drive whatever phototypesetter the customer had. In time Atex was delivering a full editorial system including digital images and had many different systems in its product line (Figure 5).[19]

Atex 8000 (PDP-11/05 or 11/35, or 04 and 34).
Computer also serves as controller for Atex-built terminals.
9000 newspaper system.
7000 newspaper system (April 1981).
Models 7032, 7048 and 7064.
4000 (see also AKI).
4000S (with Release 4 software) 1984.
GT68 Graphics and pagination terminal.
(Motorola 68000) (1982).
Classified ad pagination.
Release 3 introduced 1979.
Atex PC interface software supports virtually any PC running virtually any word processing and communication software.
Release 4 software (1981).
Spellcheck (1982).
5000 newspaper system (1982).
Atex 1000 remote terminal (1981).
Atex 1500 "full-function" remote (1981).
Atex 9080 remote cluster (1981).
Atex 500 remote terminal (1982).
Atex library or morgue system.
Electronic library system jointly with Infotex (1981).
Acquired by Eastman Kodak 1981.
TPE (Total Publishing Environment) products.

Figure 5: Atex product line circa 1985

Naturally competition for the Atex systems developed and managing the business became harder as the company became bigger. Thus, in 1981 the founders sold the company to Kodak. Charles Ying remained with the company until Kodak closed the part of the company for which he was working. After Atex, he stayed close to the publishing industry, for instance serving at different times as president of Information International and Bitstream.[20]

3 Desktop publishing

Today desktop publishing is everywhere. However, in 1980, commercial typesetting for newspapers, periodicals, and books was still a separate domain, and commercial word processing products were a relatively new product, mainly not thought of as a tool for publishing.

3.1 Xerox PARC

The commercial desktop publishing market developed over the 1980s. However, much of the technology enabling what we now think of as desktop publishing was developed in the 1970s. Of course, computing and electronics technology had been improving for a long time with the work of many companies and people. But a surprising amount of the relevant technology was demonstrated by Xerox Corporation, particularly at Xerox Palo Alto Research Center (PARC).

As a research organization in Xerox, PARC did development that (nominally) related to computerization of the office. Below is a list of some of what they developed — at least prototypes and sometimes distributed fairly widely within Xerox or outside the company.[21,22,23,24,25]

- The Alto networked (via Ethernet) workstation (1973) with a raster display providing a graphical user interface.
- Laser xerographic printers that could print high resolution bitmaps for output pages.
- Printer servers (Electronic Array Raster Scanner) on the local area network.
- "Press files" that could intermingle text and graphics.
- The Fred program to create (on the Alto) outline fonts for printing and display using cubic splines.
- The Draw program to create figures made up of text, lines, and curves, again using cubic splines.
- The Press program to print Press files.
- The Bravo and Gypsy WYSIWYG editors.
- Interpress page description language (the predecessor of PostScript).

The above technology was known outside of PARC and thus aspects of it were highly influential as the desktop publishing world developed. It also fed into the word processing world.

3.2 Seybold Reports and Seminars

I began the story of the Seybolds' activities in the publishing world (starting in 1963) in the Rocappi section (section 2.3). Another way the world got ready for desktop publishing was through the activities of the Seybolds throughout the 1970s, during which they found ways to keep current about and to push forward publishing technology.

The section's sketch is taken from Jonathan Seybold's paper about the Seybold Reports and Seybold Seminars.[26]

Seybold Reports

After leaving Rocappi, John Seybold remained on the east coast and established a company, John W. Seybold & Associates, to consult to publishing companies interested in applying computing technology. Jonathan Seybold moved to southern California. He initially helped Autologic company by specifying the typographic capabilities of their new APS-4 phototypesetting system. By the spring of 1971, John and Jonathan had begun to discuss writing another book but decided the publishing world was moving too fast for a book. A better idea would be a bi-monthly "newsletter", except that it would only contain in-depth analysis.

The first year, the Graphic Communications Computer Association (GCCA) operated the newsletter, which they insisted would be called *The Seybold Report*, while the Seybolds provided all the content, typically a single long article on one product or product line or occasionally a tutorial on important technology or market trends. The Seybolds took great care to be accurate and conflict free, while producing issues of 12–16 pages of typewriter copy. About six months in, they added some pages of news at the back of each issue, so there was something of interest to readers not interested in the feature report in the issue. Industry trade shows provided good sources of news.

In the second year the Seybolds took over the business of the Report themselves, with John handling the business and Jonathan handling the "intellectual side". This was the start of Seybold Publications. The Report then expanded to 20–24 phototypeset pages. By the end of the second year, 25 percent of subscribers were from outside the U.S.

John and Jonathan (as Seybold Publications) also gave two-day tutorials several times a year on latest developments in the industry. These were arranged in the U.S. by GCCA and by the Printing Industry Research Association in the U.K. As with the trade shows, the tutorials were an opportunity to meet and get to know more people, thus building their network and knowledge. Between them they also continued the consulting work they had begun before starting the Report, for little newspaper groups, big technology companies, and big publishing companies.

While the Report's subscription base and other aspects of the Report grew, it never made much money. Without advertising, the report was priced in the hundreds of dollars which limited potential subscribers and encouraged reading of a subscription by multiple people and even piracy. However, Jonathan says,

> The Report was highly successful in achieving its primary objective: The technological base of an entire industry was being re-made in a single decade. ...
>
> We were right in the middle of all of this. I like to think that we played an important and constructive role in helping to shape how it all came out.

By the early 1980s, Jonathan's sister Patricia had started the *Seybold Report on Word Processing*, and a little later Jonathan and Patricia started a report on personal computers, and to sort this all out, they renamed the reports, i.e.,

- *The Seybold Report on Publishing Systems* — the original Seybold report
- *The Seybold Report on Office Systems* — prior report on word processing
- *The Seybold Report on Professional Computing* — the PC report

When John Seybold eventually retired, Jonathan kept the publishing report, Patricia kept the office systems report, and second son Andrew took the computing report.

John Seybold had been "a true pioneer in automated typesetting."[27] Frank Romano, in his dedication to his book on the phototypesetting era continues, saying of Seybold: Rocappi served "as the world's first commercial computer typesetting service bureau." Seybold published books on "the new typesetting machines, companies, concepts, and applications", and that Seybold was first to apply "what you see is what you get" to "display screen applications". "He played a key role in the decision by *U.S. News and World Report* to become the first customer for the Atex Publishing System." And with Jonathan, John created the *Seybold Report*, which (page 296 of Romano's book) John called, "'a book that had to be constantly updated' and promised to cover every photocomposition and text editing device on the market", and "they tested every system and reported their results and critiques".

Seybold Seminars

As Jonathan saw the coming world of what became desktop computing, he started the Seybold Seminars as a way ...

> ... to get the people involved together for a conference designed to encourage interchange.
>
> Four $1\frac{1}{2}$ hour sessions per day with generous time for a group lunch and generous morning and afternoon breaks to encourage lots of informal interaction. Two presentations per session. No sales pitches.

The seminar was an annual event. In keeping with Jonathan's goal, it became a place where developers of desktop publishing systems and other relevant parties got together.

From near the beginning, some seminar attendees wanted to bring equipment to show. In 1986 Jonathan put together a desktop publishing conference that included a trade show. He also launched a new *Seybold Report on Desktop Publishing* at the same time. In time, the original conference also grew into a trade show, with one held annually in San Francisco and the other (a little smaller) held in Boston. Both combined the conference with the trade show, with the former having a few thousand attendees and the latter having tens of thousands of attendees. Also, as the technology evolved, the distinction between professional publishing and desktop publishing disappeared. The Seybold Seminars continued to expand.

Seybold and his people had always helped the press when asked for answers or pointers to other people. This was consistent with their mission to help the industry change happen. (Being quoted in the press was also good PR.) In time they established an explicit press liaison office to help the press. The same staff members also helped the PR people in the companies they dealt with who might be inexperienced and need pointers within the industry.

In 1989 Jonathan established a *Digital World* conference, independent of the publishing conferences, for people interested in the ever increasingly digital world, and the monthly *Digital Media* publication came next.

In 1990 Jonathan sold the Seybold Seminars and Seybold Publications to Ziff, while continuing to work on these activities for the next few years. He then left. Of this he says,

> For me, it had been a great ride for a quarter century. I was able to play a role in three successive revolutions: the computerization of the print publishing industry, the democratization of publishing (desktop publishing), and then helping a little to lay the foundations for our current Digital World.

On pages 296 and 297 of his book,[6] Frank Romano vouches for Jonathan's role in helping create the revolution discussed in the next section.

3.3 Commercial desktop publishing

The systems discussed in this section are what we now call desktop publishing systems — DTPs. For many people, the definition of DTP involves a WYSIWYG interface running on a desktop computer. To my mind, the work stations from Sun, Apollo, etc., on which some of the systems in this section initially ran were the early versions of today's personal computers.

The companies touched upon in the following sketch of creation of the desktop publishing business are Adobe,[28,29,30] Aldus,[31,32] Apple,[33] Frame Technology,[34,35] Interleaf,[36,37] Quark,[38] and Ventura.[39] There were also other companies that I have omitted.

In 1981 the Seybold Seminars (as described in the prior section) had been started as a way for people to get together who might advance the use of digital technology for publishing. Many of the involved people also subscribed to the *Seybold Report* or were reading issues from other people's subscriptions. Seybold was constantly scouting what was happening in the publishing and publishing technology industry for his reports and seminars; he tended to know what everyone was doing. As another point of reference, by 1980 outline fonts were available to publishers from Linotype and Compugraphic, but bitmapped fonts were still typically used for laser printing, screen display, and in the newspaper industry.

Adobe was founded in 1983 to push the vision of Charles Geschke and John Warnock for a page description language that Xerox PARC had not been interested in pursuing, and in 1983 they were able to demonstrate a prototype PostScript laser printer. In 1984 Adobe released Level 1 PostScript. Along with this came Adobe's Type 1 and Type 3 fonts. Adobe also did a deal to use ITC fonts in PostScript. With PostScript, outline fonts began to spread for laser printing and screen display.

Also in 1984, an Apple Mac computer with a graphical user interface was available. Steve Jobs had excitedly shown Jonathan Seybold a Mac the year before, and in 1983 Jobs called Seybold back to Apple to show him a Mac connected to a LaserWriter with built-in PostScript. Apple and Adobe had done a deal about PostScript and raster output devices. Seybold says that the Mac-LaserWriter-PostScript combination indicated to him that a revolution in the publishing world was imminent; he also knew what Aldus Corporation was doing.

Aldus Corporation was founded in 1983 by Paul Brainerd. Out of college he worked in operations for the *Minneapolis Star and Tribune* while they converted from metal type to computer-based typesetting. Atex was a key supplier. Next Brainerd went to Atex and stayed there until it was sold. Then he started Aldus which created PageMaker, initially for the Mac. Brainerd is credited with coining the term "desktop publishing". PageMaker was aimed at small businesses and also used by professional and amateur book designers and others. Jonathan Seybold encouraged Brainerd to get together with the right group at Apple to see the Mac with its PostScript laser printer and also encouraged people at Apple to talk to Brainerd.

The deal between Apple and Adobe resulted in the 1985 product release of the LaserWriter with built-in PostScript with Adobe's Type 1 and 3 font technology. In 1985 Aldus PageMaker for "desktop publishing" also was released, and groups at Apple, Adobe, Aldus began an informal collaboration marketing desktop publishing to small businesses. Apple sold the Mac hardware, Adobe got paid for every PostScript LaserWriter that Apple sold, and Aldus sold its PageMaker software package. It was thus in everyone's interest to help each other selling this "desktop publishing solution". They had found a significant untapped market, and pushing desktop publishing was a major benefit for all three companies. (Adobe was also licensing PostScript to other printer and computer manufacturers.)

Interleaf also released a desktop publishing system in 1985 — their Interleaf Technical Publishing Software (TPS); the company had been founded in 1981. Their product was aimed at technical publishing and distribution with integrated text and graphics. It originally ran on Sun and Apollo workstations. The system was programmed in Lisp (Interleaf Lisp), and users could modify the system.[40] By 1987 Interleaf was also running on more workstations, on the Mac, and under Windows.

Also in 1986, Ventura Publisher for the IBM PC was released by Ventura Software which had been founded the previous year. The founders felt they had an innovative way (better than PageMaker) to lay out multi-article documents; their system also became the first popular desktop publishing system for the IBM PC class of computers.

Yet another desktop publishing company was founded in 1986 — Frame Technology. They released FrameMaker for the Sun and other Unix workstations. Charles Corfield had developed FrameMaker aimed at publishing large and very large and complex documents, and thus a competitor of Interleaf. (David Fuchs was the fifth employee of the company after the four founders.)

In 1987 QuarkXPress 1.0 for the Mac was released, aimed at the high-end publishing (profes-

sional typesetting and page layout) market, competing with PageMaker there. (Quark had been founded in 1981 and did other things before going into desktop publishing.) Within only two or three years, QuarkXPress was putting serious pressure on PageMaker and Aldus. Aldus brought out successive versions of PageMaker, and also broadened its product line through acquisitions of other products.

By 1989 the tiny staff of Ventura was getting tired (they never had more than five employees), and in 1990 Ventura was acquired by Xerox, which had been the distributor of Ventura Publisher from the beginning. Three years later, Xerox sold the Ventura business to Coral, which continues to sell the system today as Coral Ventura. Also in 1990, Quark brought out QuarkXPress 3.1 for Windows, and Quark became the dominant player and QuarkXPress the industry standard in the market.

By 1993 PageMaker had lost considerable market share to Quark and other desktop publishing systems. With such increasing competition for PageMaker and Aldus' other products not generating enough sales for the company to continue its early extraordinary growth and profit, in 1993 Paul Brainerd initiated talks with Adobe and in 1994 Aldus was acquired by Adobe.

Meanwhile, FrameMaker had been made to run on Unix, Macs, and Windows PCs, and the company tried to also compete in the home desktop publishing market, which was a loss of business focus leading to near insolvency. Adobe bought the product in 1995, refocused on the business market, and the product still has a significant following today.

QuarkXPress 4.0 continued the market dominance by Quark. Adobe countered by rewriting PageMaker and bringing the resulting software out as InDesign 1.0. Over time InDesign cut deeply into Quark's market, although I think there is still competition between Quark and InDesign today. InDesign is used by professional book designers and typesetters (and by amateurs who want good typesetting and would never think of using LaTeX).

In 2000 Interleaf was was acquired by Broadvision, and the product was renamed Quicksilver.

Desktop publishing tapped a large market that effectively included consumer products as well as products for professionals. A common method of document interchange, both layout and fonts, was a natural outgrowth. Adobe and PostScript won the battle for dominance over other companies and technologies. The competition for digital font technology dominance resulted in a compromise.

By 1986, Adobe had pushed PostScript into graphics applications, and a couple of years later brought out the Encapsulated PostScript (EPS) format for graphics.

In 1989 Apple and Microsoft began what was called the "font wars" when Microsoft claimed at a Seybold conference that their (unfinished) TrueType font technology was superior to PostScript font technology. Naturally, Adobe forcefully disagreed, and Adobe brought out Adobe Type Manager for Mac, Windows, and other operating systems to counteract TrueType. Apple's first release of TrueType was in 1991, and Microsoft released TrueType for Windows 3.1 in 1992. In parallel with the competition about font technology, Adobe kept pushing PostScript, bringing out PostScript Level 2 in 1991, and Acrobat and Portable Document Format (PDF) in 1993. With OpenType in 1996, Adobe, Apple, and Microsoft combined their competing font technology approaches.

I have heard PostScript being described as a page description language or as a language for creating vector graphics. It was originally aimed at driving printers and first became well known by its use in Apple's computers. PostScript (and EPS and PDF) have clearly changed the way the typesetting and printing worlds work.

In about a dozen years, the desktop publishing market had developed and then consolidated.

Next step and acknowledgments

I was unable to attend TUG 2018 and present this content there, and this paper will have to do. I do intend to finish the monograph I mentioned at the beginning of this paper. The monograph will include relevant parts of my TUG 2012 and TUG 2016 papers, the content of this paper, and a bit more.[41] It will be posted at `tug.org/l/walden-digitype-monograph`.

Over the past half a dozen years, dozens of people have answered questions about the topic of this paper (and its 2016 predecessor) or otherwise helped me with the paper(s). I greatly appreciate the help of each of them.

I must specifically acknowledge Burt Grad who invited me to participate in the May 2017 desktop publishing pioneers meeting at the Computer History Museum.

In the past several years, I have also had more or less frequent contact on a variety of these topics with Barbara Beeton, Karl Berry, Chuck Bigelow, David Hemmendinger, and Jonathan Seybold. Each has contributed to my education on the history of digital typography.

References and notes

[1] walden-family.com/bbf/bbf-printing.pdf

[2] tug.org/tug2016/walden-digital.pdf

[3] history.computer.org/annals/dtp

[4] Simonyi stopped by the meeting but didn't participate in any of the sessions.

[5] John W. Seybold, *The World of Digital Typesetting*, Seybold Publications, Inc., Media, PA, 1984, computerhistory.org/collections/catalog/102740425.

[6] Frank Romano, *History of the Phototypesetting Era*, Graphic Communications Institute at Cal Poly State University, 2014.

[7] Mike Marcus and George Trimble, Taking Newspapers from Hot Lead into the Electronic Age, *IEEE Annals of the History of Computing* vol. 28 no. 4, 2006, pp. 96–100.

[8] Jonathan Seybold, Early steps in computer typesetting in the 1960s, September 2018, history.computer.org/annals/dtp/rocappi-typesetting.pdf.

[9] en.wikipedia.org/wiki/Michael_P._Barnett

[10] Michael P. Barnett, *Computer Typesetting: Experiments and Prospects*, MIT Press, 1965.

[11] Jonathan W. Seybold, Rocappi: Computerizing the Publishing Industry, *IEEE Annals of the History of Computing*, vol. 40 no. 3, 2018, pp. 8–24.

[12] Jonathan, having had his first experience with a computer as a student of economics in 1964, joined his father's company in 1965 where within a year he was effectively running Rocappi's production operations.

[13] Jonathan Seybold, email of 2019-05-08.

[14] Rowley Atterbury, Colin Barber: Computer pioneer who launched the greatest printing revolution since Gutenberg, *The Guardian* obituary, October 3, 2006, theguardian.com/technology/2006/oct/04/news.guardianobituaries

[15] Jonathan Seybold, email of 2019-05-13.

[16] John Pierson, *Computer Composition Using PAGE-1*, John Wiley & Sons, 1972.

[17] Douglas Drane, How Atex Helped an Industry Change the World, *IEEE Annals of the History of Computing*, vol. 40 no. 3, 2018, pp. 25–29; Jonathan Seybold and David Walden, More about Atex, *IEEE Annals of the History of Computing*, vol. 40 no. 3, 2018, pp. 30–36.

[18] Wilson R. Locke, Telecommunication in the News Industry: The Newsroom Before and After Computers, a chapter in *People and Technology in the Work Place*, Natl. Academy of Engineering, Washington, D.C., 1991.

[19] The figure is taken from the 1985 supplement of John Seybold's book, reference 5 above.

[20] Andrew Tribute, Charlie Ying 1946–2010, *What They Th!nk*, October 20, 2010, tug.org/l/ying-obit

[21] Gardner Hendrie, interview of Gary Starkweather, Computer History Museum, 2010, tug.org/l/chm-starkweather

[22] Robert F. Sproull, The Xerox Alto Publishing Platform, *IEEE Annals of the History of Computing*, vol. 40 no. 3, 2018, pp. 38–54.

[23] John E. Warnock, The Origins of PostScript, *IEEE Annals of the History of Computing*, vol. 40 no. 3, 2018, pp. 68–76.

[24] Lawrence G. Tesler, How Modeless Editing Came To Be, *IEEE Annals of the History of Computing*, vol. 40 no. 3, 2018, pp. 55–67.

[25] Robert F. Sproull, Publishing a Computer Graphics Book With Prototype Desktop Publishing Tools, *IEEE Annals of the History of Computing*, vol. 40 no. 4, 2018, pp. 69–76.

[26] Jonathan W. Seybold, Seybold Publications and Seminars, *IEEE Annals of the History of Computing*, vol. 41 no. 3, 2019.

[27] Reference 6, dedication page.

[28] John E. Warnock, The Origins of PostScript, *IEEE Annals of the History of Computing*, vol. 40 no. 3, 2018, pp. 68–76.

[29] Pamela Pfiffner, *Inside the Publishing Revolution: The Adobe Story*, Peachpit Press, Berkeley, CA, 2003.

[30] John Warnock and Charles Geschke, Founding and Growing Adobe Systems Inc., *IEEE Annals of the History of Computing*, vol. 41 no. 3, 2019.

[31] Paul Brainerd oral history: tug.org/l/brainerd-oralhistory

[32] Suzanne Crocker, Paul Brainerd, Aldus Corporation and the Desktop Publishing Revolution, *IEEE Annals of the History of Computing*, vol. 41 no. 3, 2019.

[33] John Scull and Hansen Hsu, Desktop Publishing: The Killer App That Saved the Macintosh, *IEEE Annals of the History of Computing*, vol. 41 no. 3, 2019.

[34] David J. Murray, Frame Technology and FrameMaker, *IEEE Annals of the History of Computing*, vol. 41 no. 3, 2019.

[35] David Murray, The FrameMaker Document Model, history.computer.org/annals/dtp.

[36] Four Interleaf documents and three books may be found at bitsavers.org/pdf/interleaf.

[37] Paul M. English and Raman Tenneti, Interleaf Active Documents, *Electronic Publishing*, vol. 7 no. 2, June 1994, pp. 75–87, researchgate.net/publication/228057523_Interleaf_Active_Documents.

[38] Jay Nelson, Interview with Tim Gill, *IEEE Annals of the History of Computing*, vol. 41 no. 3, 2019.

[39] Lee Lorenzen, The Ventura Story, *IEEE Annals of the History of Computing*, vol. 41 no. 3, 2019.

[40] Karl Berry was with Interleaf for a while. Some readers may remember author Tracy Kidder attending the 2014 TUG conference in Portland, Oregon. Kidder wrote a book called *A Truck Full of Money*, published in September 2016, that talks a good bit about Interleaf as a development organization and business.

[41] Charles Bigelow, Font Wars parts 1 and 2, *IEEE Annals of the History of Computing*, to be published in early 2020.

◇ David Walden
walden-family.com/texland

TEX services at texlive.info

Norbert Preining

TEX Live has grown over the years from a DVD distributed once a year to a full-fledged network-enabled system with daily updates. Development of TEX Live is done in the Subversion repository on tug.org, and coordinated on several mailing lists.[1] The network distribution of TEX Live is served by CTAN and their mirror system, relieving us from serving huge amounts of data to the world, or reimplementing another mirror system. (Thanks much to CTAN.)

Around the main distribution of TEX Live, several additional services have accumulated over the years — and in particular in recent months — so that we thought it a good idea to summarize and present all the available services in this article. Details about each follow the overview.

1 Overview of the services

- TEX historic archives via rsync
  rsync://texlive.info/historic
- tlnet archive via https
  https://texlive.info/tlnet-archive
- tlpretest mirror
  - via https
    https://texlive.info/tlpretest
  - via rsync
    rsync://texlive.info/tlpretest
- CTAN
  - mirror via https
    https://texlive.info/CTAN
  - mirror via rsync
    rsync://texlive.info/CTAN
  - git repository
    git://git.texlive.info/CTAN
  - git web interface
    https://git.texlive.info/CTAN
- TEX Live git mirror
  - repository
    git://git.texlive.info/texlive
  - web interface
    https://git.texlive.info/texlive
  - statistics
    https://texlive.info/tlstats
- TEX Live GnuPG
  - repository
    https://texlive.info/tlgpg
  - git web interface
    https://git.texlive.info/tlgpg
  - git repository
    git://git.texlive.info/tlgpg
- TEX Live contrib
  - repository
    https://contrib.texlive.info
  - git web interface
    https://git.texlive.info/tlcontrib
  - git repository
    git://git.texlive.info/tlcontrib

For the git services, anonymous checkouts are supported. If a developer wants to have push rights, please contact me.

2 TEX Live historic archive

The TEX Live historic archive[2] hierarchy contains many items of interest in TEX history, from individual files to entire systems. See the *TUGboat* article by Ulrik Vieth [1] for an overview.

We provide a mirror, available via rsync:
rsync://texlive.info/historic

3 tlnet archive

TEX Live is distributed via the CTAN network at http://mirror.ctan.org/systems/texlive/tlnet. The packages there are updated on a daily basis, mostly stemming from uploads to CTAN that are imported into the TEX Live repository. This has created some problems for distributions requiring specific versions, as well as problems with rollbacks in case of buggy packages.

As of 2019-08-30, we archive (by date) the daily tlnet updates, available at https://texlive.info/tlnet-archive. We have also added the final tlnet for TEX Live releases going back about ten years.

4 tlpretest mirror

During preparation of a new TEX Live release (the pretest phase[3]) we are distributing preliminary builds via a few tlpretest mirrors.[4] The present server will also provide access to tlpretest:

- via https
  https://texlive.info/tlpretest
- via rsync
  rsync://texlive.info/tlpretest

5 CTAN-related services

Besides providing another mirror for CTAN, we have taken steps to insert the continual updates at CTAN into a git repository. In a perfect world we could get separate git commits for each package update, but that requires significant work from the CTAN team (maybe this will happen in the future); so for now there is one commit per day containing all changes.

[1] https://tug.org/texlive/lists.html

[2] https://tug.org/historic

[3] https://tug.org/texlive/pretest.html

[4] https://tug.org/texlive/mirmon

Considering the total size of CTAN (currently around 40GB), we decided to ignore files of types that provide no useful information when put into git, namely large binary files. The concrete list is: `cab deb dmg exe iso jar pkg rpm tar tgz zip`, including names containing one of these extensions (meaning that files like `foo.iso.gz` will be ignored too). This keeps the size of the `.git` directory at something reasonable (a few GB for now).

We will see how the git repository grows over time, and whether we can support this permanently.

While we exclude the above large files from being recorded in the git repository, the actual CTAN directory is complete and contains all files, so everything is available through rsync or https.

Access to these services is provided as follows:

- mirror via https
  `https://texlive.info/CTAN`
- mirror via rsync
  `rsync://texlive.info/CTAN`
- git repository
  `git://git.texlive.info/CTAN`
- git web interface:
  `https://git.texlive.info/CTAN`

6 TEX Live svn/git mirror

Since I prefer to work with git, and developing new features with git on separate branches is much more convenient than working with Subversion, I am running a git–svn mirror of the whole TEX Live subversion repository.[5] It is updated every 15 minutes.

There are also git branches matching the subversion branches, and also some `dev/` branches where I am working on new features. The git repository carries, similar to the subversion, the full history back to our switch from Perforce to Subversion in 2005. This repository is quite large, so don't do a casual checkout — the size is currently close to 40GB.

We also (at irregular intervals) compute some statistics of this repository using gitstats.[6]

- git repository
  `git://git.texlive.info/texlive`
- git web interface
  `https://git.texlive.info/texlive`
- statistics
  `https://texlive.info/tlstats`

(Incidentally, the TEX Live repository on github, `https://github.com/TeX-Live/texlive-source`, contains only the relatively small tree of source files to be compiled, due to github limitations.)

[5] `https://tug.org/texlive/svn`
[6] `http://gitstats.sourceforge.net`

7 TEX Live GnuPG

Starting with the 2016 release, TEX Live provides facilities to verify authenticity of the TEX Live database using cryptographic signatures. This requires a working GnuPG program, either `gpg` (version 1) or `gpg2` (version 2). To ease adoption of verification, this repository provides a TEX Live package `tlgpg` that ships GnuPG binaries for Windows and MacOS (`universal` and `x86_64`). On other systems we expect GnuPG to be installed.

- repository
  `https://texlive.info/tlgpg`
- git web interface
  `https://git.texlive.info/tlgpg`
- git repository
  `git://git.texlive.info/tlgpg`

8 TEX Live contrib

The TEX Live Contrib repository[7] is a companion to the core TEX Live (tlnet) distribution in much the same way as Debian's non-free tree is a companion to the normal distribution. The goal is not to replace TEX Live — packages that could go into TEX Live itself should stay (or be added) there. The TEX Live Contrib tries to fill a gap for users by providing ready made packages for software that cannot be distributed in TEX Live proper due to license reasons, support for non-free software, etc.:

- repository
  `https://contrib.texlive.info`
- git web interface
  `https://git.texlive.info/tlcontrib`
- git repository
  `git://git.texlive.info/tlcontrib`

9 Supporting these services

We will try to keep this service up and running as long as server space, connectivity, and bandwidth allow. If you find them useful, I happily accept donations via PayPal[8] or Patreon[9] to support the server as well as my time and energy!

◇ Norbert Preining
Accelia Inc., Tokyo, Japan
`preining.info`

References

[1] U. Vieth. Overview of the TEX historic archive. *TUGboat* 29:1 (2008), pp. 73–76.
`https://tug.org/TUGboat/tb29-1/tb91vieth.pdf`

[7] `https://contrib.texlive.info`
[8] `https://tinyurl.com/preining-paypal`
[9] `https://www.patreon.com/norbert`

Providing Docker images for TEX Live and ConTEXt

Island of TEX

Abstract

With the spread of version control and continuous integration services among TEX users there is a need to provide TEX distributions for containerized services. As most available images are not updated regularly and many of them lack relevant tools, we aim to provide images for the regular user who wants continuous integration to work like any other TEX distro.

1 An excursion into continuous integration and the relevance of Docker images

Today, many TEX users rely on version control to have a steady backup of their document sources. While the sources are handled smoothly by version control systems, binary files such as PDF are not. To prevent bloating the repository with such, it is very useful to have an alternative to pushing a compiled result and still have it available on request. That is where continuous integration steps in.

Basically, this requires the user to add yet another text file to your repository specifying how the continuous integration service (CI) should handle the document. As an example we will have a look at a GitLab CI file. (The `image:` line is only broken for *TUGboat*; it should be all one line in the source.)

```
image: registry.gitlab.com/islandoftex/
images/texlive:latest

build:
  script:
    - arara -v mydocument.tex
  artifacts:
    paths:
      - ./*.pdf
```

The above `.gitlab-ci.yml` file tells the GitLab CI to pull our latest image of TEX Live (without documentation and source files, as discussed later on) and execute one build stage: calling `arara` on a file `mydocument.tex`. After the run has finished all PDF files from the current folder will be saved as artifacts and are available for download.

Above, the first line represents the Docker image the CI will use; most providers offer a similar way to specify the image. A Docker image itself is similar to a snapshot of a lightweight virtual machine. When running an image (then it is called a container, broadly speaking) 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.

2 Using the TEX Live images

For most (LA)TEX documents out there, people will want to use a complete TEX distribution. Hence, we are providing multiple images of TEX Live. The respective Dockerfiles can be found at `https://gitlab.com/islandoftex/images/texlive`.

The most important image is `texlive:latest`. This image is based on GNU/Linux and ships with all required tools for running TEX Live, among them Java (e.g. for arara), Python (e.g. for Pygmentize) and Perl (e.g. for xindy).

Please note that the `latest` image does not contain the documentation and source tree of TEX Live. For users that need these components we provide `latest-doc`, `latest-src` and `latest-doc-src` which contain documentation, sources or documentation and sources respectively.

The images starting with `latest` are rebuilt weekly. If you want to use a stable snapshot, you can select the one that suits your needs at `https://gitlab.com/islandoftex/images/texlive/container_registry`.

To use our images in a custom Dockerfile, you can use the following line (again broken only for *TUGboat*):

```
FROM registry.gitlab.com/islandoftex/
images/texlive:latest
```

The source repository already contains Dockerfiles for providing historic releases. While you have to build them yourself for now, we are confident to provide the images in the near future.

3 Using the ConTEXt images

Similar to the TEX Live images, we provide images with the ConTEXt standalone distribution. The respective Dockerfiles can be found at `https://gitlab.com/islandoftex/images/context`.

Apart from the MkIV current release `current` there are also images for the MkIV beta release `beta` and the LMTX release `lmtx`. For the available snapshot releases, go to `https://gitlab.com/islandoftex/images/context/container_registry`.

The use in custom Dockerfiles is similar to the TEX Live images:

```
FROM registry.gitlab.com/islandoftex/
images/context:lmtx
```

◇ Island of TEX (developers)
`https://gitlab.com/islandoftex`

TEX on the Raspberry Pi

Hans Hagen

This is a short status report on Pi, not the famous version number of TEX (among other things), but the small machine, meant for education but nowadays also used for Internet Of Things projects, process control and toy projects. While the majority of TEX installations run on an Intel processor, the Raspberry Pi has an ARM central processing unit. In fact, its main chip has the same foundation as those found in settop boxes all around the world. It's made for entertainment, not for number crunching.

At the ConTEXt meetings, it has become tradition to play with electronic gadgets. Every year we are curious what Harald König might bring this time. The last couple of meetings we also had talks about using TEX and MetaPost for designing (home-scale, automated) railroad systems, using LuaTEX for running domotica applications, using MetaPost for rendering high quality graphics from data from appliances, presenting TEX at computer and electronics bootcamps, and more. This year Frans Goddijn also brought back memories of low speed modem sounds, from the early days of TEX support. It is these things that make the meetings fun.

This year the meeting was in Belgium, close to the border of the Netherlands, and on the way there Mojca Miklavec traveled via my home, where the contextgarden compile farm runs on a server with plenty of cores, lots of memory and big disks. But the farm also has an old Mac connected as well as a tiny underpowered Raspberry Pi 2 for ARM binaries that we had to fix: the small micro SSD card in it had finally given up. This is no surprise if you realize that it does a daily compilation of the whole TEX Live setup and also compiles LuaTEX, LuaMetaTEX and pplib when changes occur. Replacing the card worked out but nevertheless we decided to take the small machine with us to the meeting. We also took an external (2.5 inch) SSD box with us. The idea was to order a Raspberry Pi 4 on location, the much praised successor of the older models, the one with 4 GB of memory, real USB 3 ports and proper Ethernet.

At the meeting Harald showed us that he had version 1, 3 and 4 machines with him because he was looking into an energy control setup based on Zigbee devices. So we had the full range of Pi's there to play with.

This is a long introduction but the message is that we are dealing with a small but popular device with up to now four generations, using an architecture supported in TEX distributions. So how does that relate to ConTEXt? One of the reasons for LuaMetaTEX going lean and mean is that computers are no longer getting much faster and 'multiple small' energy-wise has more appeal than 'one large'. So then the question is: how can we make TEX run fast on small instead of gambling on big becoming even bigger (which does not seem to be happening anyway).

At the meeting Harald gave a talk "Which Raspberry Pi is the best for ConTEXt?" and I will use his data to give an overview: see Table 1.

After some discussion at the presentation we decided to discard the (absurd) bogomips value for the tiny Pi 1 computing board and not take the values for the others too seriously. But it will be clear that, especially when we consider the external drive that things have improved. The table doesn't mention Ethernet speed but because the 4 now has real support for it (instead of sharing the USB bus) we get close to 1 GB/s there.

The real performance test is of course processing a TEX document and what better to test than *The TEXbook*. The processing time in seconds, after initial caching of files and fonts is:

Pi model:	1	2	3	4
The TEXbook	13.649	7.023	4.553	1.694
`context --make`		19.949	11.796	6.034
`context --make` TL	89.454	46.578	29.256	14.146

Table 1: Capacities for Raspberry Pi models.

Pi model:	1	2	3	4
chipset	BCM2835	BCM2835	BCM2835	BCM2835
CPU core	v6l rev 7	v7l rev 5	v7l rev 4	v7l rev 3
cores	1	4	4	4
free mem	443080	948308	948304	3999784
idlemips	997.08	38.40	38.40	108.00
bogomips	997.08	57.60	76.80	270.00
read SD	23.0 MB/s	23.2 MB/s	23.2 MB/s	45.1 MB/s
read USB		30.0 MB/s	30.0 MB/s	320.0 MB/s

The test of making the ConTEXt format using LuaTEX gives an indication of how well the I/O performs: it loads the file database, some 460 Lua modules and 355 TEX source files. On my laptop with Intel i7-3840QM with 16GB memory and decent SSD it takes 3.5 seconds (and 1 second less for LuaMetaTEX because there we don't compress the format file). Somehow a regular TEX Live installation performs much worse than the one from the contextgarden.

We didn't test real ConTEXt documents at the meeting but when I came home the Pi 4 was bound again to the compile farm. Harald and Mojca had prepared the machine to boot from the internal micro SSD and use the external disk for the rest. So, when we could compile LuaMetaTEX again, I made an ARM installer for LMTX, and after that could not resist doing a simple test. First of course came generating the format. It took 6.3 seconds to make one, which is a bit more than Harald measured. I see a hiccup at the end so I guess that it has to do with the (external) disk or maybe there is some throttling going on because the machine sits on top of a (warm) server.

More interesting was testing a real document: the upcoming LuaMetaTEX manual. It has 226 pages, uses 21 font files, processes 225 MetaPost graphics, and in order to get it LuaMetaTEX does more than 50% of the work in Lua, including all font and backend-related operations. On my laptop it needs 9.5 seconds and on the Pi 4 it uses 33 seconds. Of course, if I take a more modern machine than this 8-year-old workhorse, I probably need half the time, but still the performance of the Raspberry Pi 4 is quite impressive. It uses hardly any energy and can probably compete rather well with a virtual machine on a heavily loaded machine. It means that when we ever have to upgrade the server, I can consider replacement by an Ethernet switch, with power over Ethernet, connected to a bunch of small Raspberries, also because normally one would connect to some shared storage medium.

Because I was curious how the dedicated small Fitlet that I use for controlling my lights and heating performs I also processed the manual there. After making the format, which takes 6 seconds, processing the manual took a little less than 30 seconds. In that respect it performs the same as a Raspberry Pi 4. But, inside that small (way more expensive) computer is a dual core AMD A10 Micro-6700T APU (with AMD Radeon R6 Graphics), running a recent 64-bit Ubuntu. It does some 2400 bogomips (compare that to the values of the Pi). I was a bit surprised that it didn't outperform the Raspberry because the (fast SSD) disk is connected to the main board and it has more memory and horsepower. It might be that in the end an ARM processor is simply better suited for the kind of byte juggling that TEX does, where special CPU features and multiple cores don't contribute much. It definitely demonstrates that we cannot neglect this platform.

◇ Hans Hagen
http://pragma-ade.com

MuPDF tools

Taco Hoekwater

Abstract

The application MuPDF (`http://mupdf.com`) is a very fast, portable, open-source PDF previewer and development toolkit actively supported by Artifex, the creators of Ghostscript (`http://artifex.com`).

But MuPDF is not *just* a very fast, portable, open-source PDF previewer and toolkit. It also comes with a handy collection of command-line tools that are easily overlooked.

The command-line tools allow you to annotate, edit, and convert documents to other formats such as HTML, SVG, PDF, and PNG. You can also write scripts to manipulate documents using JavaScript.

This small paper gives a quick overview of the possibilities.

1 Introduction

In recent versions of the MuPDF distribution, most of the tools have been combined into a single front-end program called `mutool`. This combines the functionality of the half dozen or so programs from earlier releases. If you use an older version of MuPDF, there will be little programs like `muclean`, but in the new combined version that functionality is now available as `mutool clean`. A similar command-line adjustment is needed for the other old command-line tool names.

In the following, I am using MuPDF 1.14.0. While the functionality of the separate tools remains roughly the same, not all versions of MuPDF have the exact same options. If you want to know the options that 'your' version of a command supports, just key in the name without arguments. Running `mutool` alone will provide a list of all known tools, and, e.g., `mutool clean` will show the list of options specific to the `clean` tool.

2 mutool clean — rewrite PDF file

If you are familiar with the general structure of PDF documents and you often work with PDF documents handed to you from other sources, this is probably the most valuable of all the tools.

Its main purpose is to 'clean up' a PDF. It can perform garbage collection on unused objects and clean up the page streams.

`mutool clean` can also convert a PDF into (near) ASCII by decompressing all the internal structures. The output is still a valid PDF, but since it has very little to no binary data any more, it can easily be inspected and possibly edited in a regular text editor. Beware though: this tends to make the file larger.

Alternatively, `mutool clean` can also convert a PDF into 'linearized' format for distribution on the web.

3 mutool convert — convert document

As the name suggests, this tool can convert a PDF file into a variety of different formats. Noteworthy supported formats in version 1.14 are: PNG, PNM, PCL, PS, PDF, SVG, HTML and plain text. Each of these has a number of sub-options to control the output format.

This is quite similar to the `convert` command from ImageMagick, except you do not need to have Ghostscript installed; it is generally faster, and uses less memory. On the down side, there are fewer output formats and options supported.

4 mutool create — create PDF document

With `mutool create`, you can create a PDF from text snippets that specify a page content stream. Each of these snippets becomes a page in the output PDF. It parses some special comments inside of those snippets to define images and fonts and page size, so for example a snippet could look like this:

```
%%MediaBox 0 0 300 300
%%Image Im0 /Users/taco/Downloads/22843.png
% Draw an image.
q
200 0 0 200 50 50 cm
/Im0 Do
Q
```

and the result would be a one-page PDF with that image centered in the page.

Because you have to write the page content stream, this is not a tool for beginners in PDF. Nevertheless, it is much easier to create a PDF this way than to write the PDF completely from scratch, because the required PDF objects and object references are generated by `mutool create`. Nevertheless, just using TEX is easier (albeit not as fast).

5 mutool draw — convert document

This is like `mutool convert`, except that it has more difficult to use options and uses a different syntax for those options. It is better to first see whether `mutool convert` can do what you want and only if it cannot, then look at `mutool draw`.

6 mutool trace — trace device calls

Produces a debug dump of the PDF document as an XML file. This can be useful to track what the

First published in *MAPS* 49 (2019.1), pp. 39–40. Reprinted with permission.

MuPDF library is actually doing, but too much information is lost from the PDF to do much else (at least, that is my experience so far).

7 mutool extract — extract font and image resources

Extracts images and embedded font resources from a PDF document, dumping them as separate files in the current directory.

8 mutool info — show information about PDF resources

Dumps detailed information about various PDF document internals to the standard output.

9 mutool merge — merge pages from multiple PDF sources into a new PDF

Combines one or more PDF documents (or pages from them) into a new combined PDF document. The fact that it can combine 'one' PDF means that this is an easy way to extract pages from a PDF. In fact, this is what I use to generate the separate article files for the on-line version of the Maps.

10 mutool pages — show information about PDF pages

In particular, `mutool pages` shows the various bounding boxes for all of the pages in XML format on the standard output.

11 mutool portfolio — manipulate PDF portfolios

PDF portfolios are a way of putting multiple independent PDFs into a single container PDF. With `mutool portfolio` you can create or modify the contents of such a portfolio.

12 mutool poster — split large page into many tiles

Splits each of the pages inside a PDF into tiles that then become the separate pages of the output PDF. It does not alter the page streams; `mutool poster` just creates adjusted bounding boxes for those separate pages so that they offer a different viewport to the original content stream.

This is a useful trick if you want to print a large PDF on a desktop printer, but it does not offer any extra features like registration marks.

13 mutool sign — manipulate PDF digital signatures

For now (this is a very new tool) this does nothing except verify an existing signature in a signed PDF document. And mine does not even do that yet, because it seems to be a build option that is not turned on in the MacOS version...

14 mutool run — run JavaScript

The MuPDF library has a quite elaborate interface to JavaScript that can be used with `mutool run` to execute scripts. These scripts have direct access to the command line and the MuPDF internals both for interpreting and for creating PDF documents, so very powerful things can be done.

The full interface is documented on the Artifex web pages.

15 mutool show — show internal PDF objects

`mutool show` is a tool for displaying the internal objects in a PDF file on the standard output. This can be very useful in combination with 'grep' for example, or if you do not want to load a multi-megabyte PDF in a text editor.

◇ Taco Hoekwater
taco (at) bittext dot nl

LaTeX on the road

Piet van Oostrum

1 The context

In July 2018, my wife Cary and I were travelling in South America to visit friends in Brazil and Bolivia, and additionally to have some vacation. We wanted to travel light, so I had decided not to take my MacBook with me, saving a little bit more than 2 kgs. of weight. We both had our iPhones and iPads (mine is an iPad mini), and we hoped that would do. They were mainly to be used for reading email, interactions on social media, searching for city and transport information, and the like.

I did not expect to do any TeX work, maybe some light programming, for which I had a Python system (Pythonista[1]) on my iPad.

While we were travelling in Brazil, on our way to Bolivia, I got an email from a user of the `multirow` package about a possible bug. It came with a solution which was a very simple substitution, and back home on the laptop, it would have been a few minutes to make the change, check it into the version control system, do some tests, generate a new version of the documentation, and upload the new version to CTAN.

Because this person had already made a local change, and the problem was not urgent anyway, my first reaction was: I will correct it when I am back home, which, by the way, would be some two months later. However, when we arrived in Bolivia, where we were staying a couple of weeks, the temptation to solve the problem right there became too large.

First published in *MAPS* 49 (2019.1), pp. 58–70. Reprinted with permission.

[1] `http://omz-software.com/pythonista/`

Figure 1: On our way to Brazil

Figure 2: Our trip

But what would have taken at most 10 minutes at home became a major effort without having a computer with a TeX system. In the end it took me more than two days of struggling, but with victory in the end.

If I distributed the package just as a collection of `.sty` files (there are three included), with separate documentation, the task would have been simple. I could have downloaded the package from CTAN, changed the `.sty` files with a text editor in my iPad, and uploaded them back to CTAN. It might have caused some frowning from the CTAN maintainers if the version number in the documentation would have been different from the one in the `.sty` files, but that would have been temporary anyway.

However, the package is distributed as a `.dtx` file, with a corresponding `.ins` file, and a separate PDF file containing the documentation which is generated from the `.dtx` file. The `.sty` files are also generated from the `.dtx` file with the aid of the `.ins` file. This is the standard setup for most CTAN packages. But this requires the `.dtx` and `.ins` files to be processed by LaTeX (or TeX in case of the `.ins` file). And I did not have a LaTeX distribution on my iPad.

2 What were the options?

There were in practice two solutions:

- Install a LaTeX system on my iPad.
- Use an online (cloud-based) LaTeX system.

2.1 LaTeX apps on the iPad

I found two LaTeX apps in the iOS App Store: Texpad and TeX Writer (see figure 3). Both are offline apps, i.e. you don't need an Internet connection to compile your LaTeX documents. But, on the other hand, to limit the size of the application, they don't

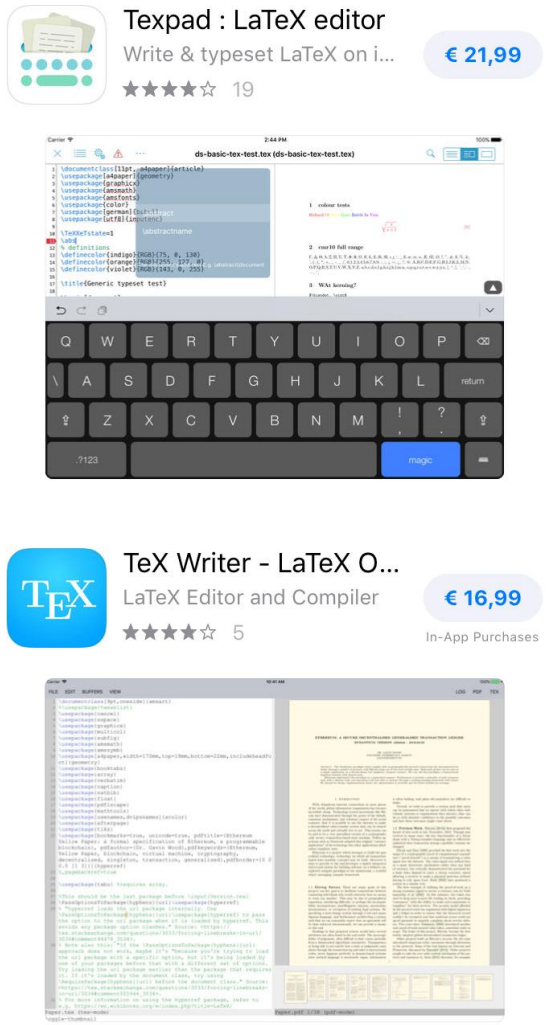


Figure 3: Texpad and TeX Writer in iOS App Store

have every package from CTAN installed. You can install additional packages, but as iOS is quite a closed operating system, you are dependent on the developers to supply these packages. Of course you can always add the required files to your project directory, but there might be some cases (e.g. if you need additional fonts) where this is not sufficient.

Also it isn't clear from the documentation of these packages if they can process something like `.dtx` and `.ins` files to extract the `.sty` files and the documentation for the package, which was essential in my case. I got the impression that they were mainly meant for the 'normal' user to write articles and reports.

They are also not particularly cheap. At the time of writing Texpad costs €21.99 and TeX Writer €16.99. If I remember correctly they were a little bit cheaper at the time I was travelling. In itself that is not a very steep price, but I did not expect to use it very often, and for just this single case I thought it was too much. And they don't have a trial version to see if it suits you, so if you buy one of these, and you don't like it, you have effectively lost your money. And then there is this nagging choice: which of the two is better? All in all, I decided not to go that way.

For the cloud-based systems, I had heard about Overleaf (formerly called WriteLatex) and ShareLaTeX, so I decided to investigate these. It appeared that at that time, these two systems were in the process of being merged. The result was Overleaf version 2 which had the ShareLaTeX interface, but was still in beta phase. For the simple task that I had, a free account would be sufficient, so I started to try that. However, the merging process introduced some teething troubles. In fact it made editing the files from the iPad browser almost impossible. It wasn't clear if this was a specific problem on the iPad, or if the browser interface in general was not yet mature enough. In effect it wasn't usable at all, because its behaviour was very erratic.

I also tried the Overleaf version 1 interface, but I could not get that working either. I have no idea whether these problems were iPad specific, but anyway I could not use it. By the way, the Overleaf editor is now functioning also on the iPad. However, some functionality is not available without an external keyboard, because they are invoked with control keys. For example the search function is invoked by Control-F on Windows and Linux, and by Command-F on MacOS. On an iPad you can't give these with the virtual keyboard. With an external keyboard it is possible. The current Overleaf editor is reasonable. It has some TEX-specific functionality. For example, if you type `\begin{enumerate}` the editor adds `\item` and `\end{enumerate}` and positions the cursor after the `\item` (see figure 4).

```
712 \begin{enumerate}
713     \item |
714 \end{enumerate}
```

Figure 4: Overleaf editor supplies useful parts

2.2 Cloud-based LATEX systems

Despite the problems that the editor gave at that time, it seemed to me that this was the best way to go forward. Figure 5 shows the screen from the current version of Overleaf on my MacBook. The default screen has an edit window with the LATEX source text and a preview window with the resulting PDF. The preview is not live, you have to hit the Recompile button to update it. There is also a file

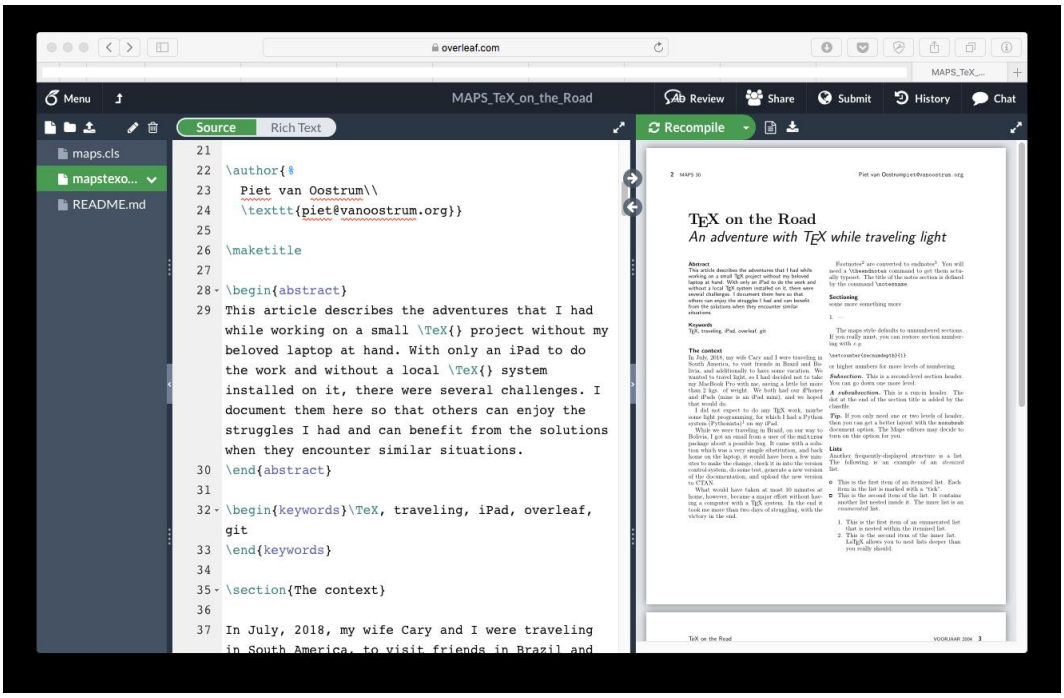


Figure 5: An early version of this article in Overleaf, with some of the `maps.cls` documentation still in place

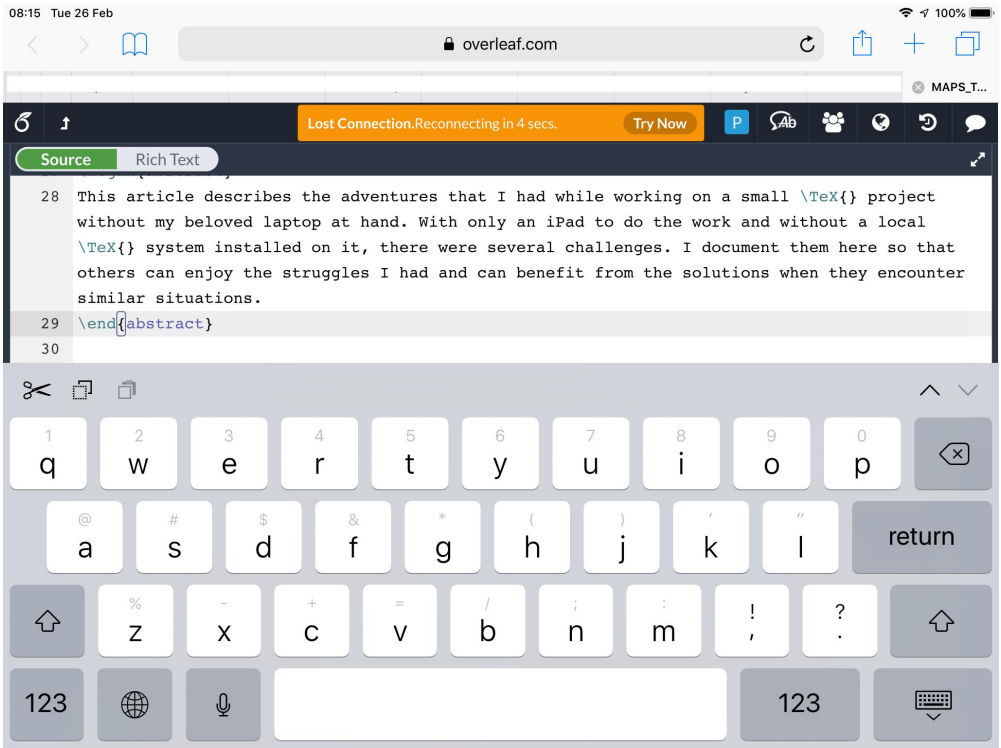


Figure 6: Overleaf screen with virtual keyboard on an iPad

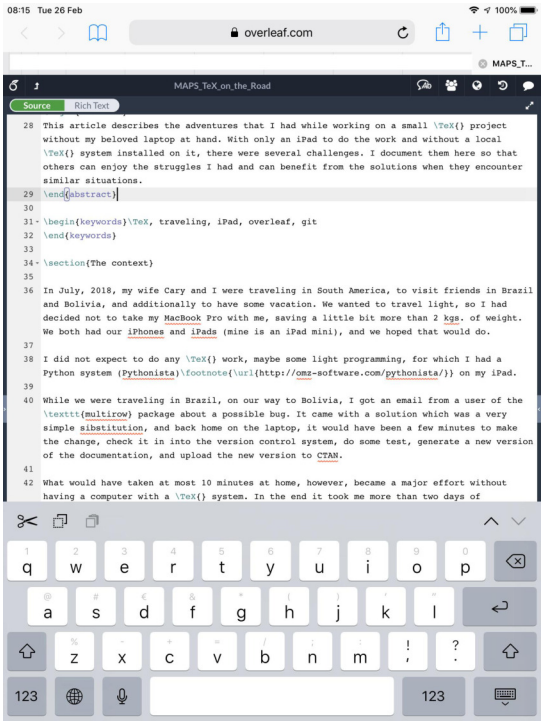


Figure 7: Overleaf screen with virtual keyboard on an iPad in portrait mode

Figure 8: iPad mini with external keyboard

list on the left and it has the capability to hide or show each of these parts and to adjust the sizes of each part. Especially on the smaller iPad screen it is advisable to have only the source code part showing while editing. But even then, the virtual iPad keyboard takes so much space that hardly any source code is visible (see figure 6). Also in this case, the file list at the left would make the edit window even smaller, but the file list can be hidden, as shown in the image.

It helps to put the iPad in portrait mode, as shown in figure 7. But then the keyboard is rather small. For a setup like this to be workable, it would be better to use an external keyboard. There are several keyboards on the market that can be used. They are generally connected through Bluetooth. They are light-weight and don't take much space, so are ideal for travelling light (see figure 8). I did not have one at that moment, however.

3 Setting up the project

Setting up the project is easy. You can create a new project in the Overleaf in the Web interface. You can upload each file individually, or a zip file with everything included. Overleaf will unpack the zip file in your project.

Immediately, it became apparent that there was a problem with my project. Overleaf wants you to designate one of your files as the main TEX file, which for me would have been `multirow.dtx`, but it doesn't accept this. It wants to have a `.tex` file. It does not recognise the `.dtx` file as a valid LATEX file. Nor does it want to edit the `.dtx` file, but as the editor was unusable, this was of a minor concern. I would have to edit the files locally on my iPad anyway.

So I had to give it a `.tex` file extension to make it (and myself) happy. I tried two ways

- Copy `multirow.dtx` to `multirow.tex`
- Make a file `multirow.tex` that just contains `\include{multirow.dtx}`

I had expected that each of these would compile the `.dtx` file when the Compile button would be pressed. However, it didn't. It took some time to find out why. My `multirow.dtx` contains a line

```
\DocInput{\jobname.dtx}
```

which is quite usual in `.dtx` files. After some searching I found out that `\jobname` wasn't `multirow` as was to be expected, but `output`. It appears that Overleaf runs the job in a kind of *sandbox* where the jobname of the main file is `output`.

After some googling I found that Overleaf uses `Latexmk`[2] to process the job. It provides a standard, but invisible, `latexmkrc` file that controls the compilation process. However, you can also supply your `latexmkrc` file. This file, and the handling of the `output` name, is described in section 'Latexmk' on page 242.

So the challenge was now to upload a correct `latexmkrc` file, and to update the `multirow.dtx` file. This could be done by uploading these files after

[2] `https://mg.readthedocs.io/latexmk.html`

each modification, but this might be an error-prone process, and you don't have a record of what has been done. Enter *version management*.

4 Distributed version management

In any project where you have to make changes more or less regularly, it is important to keep track of what you have done. Also, in general it is useful to have access to previous versions of your project, for example if you want to go back to a previous situation. Some people do this by making copies of their files at regular moments. Sometimes they put the date and the time in the file names, to keep a kind of history. But this soon becomes unwieldy. This is the problem that version management systems (also called version control systems) offer a solution for. Any serious developer, whether of software or text, should consider using a version management system.

For those readers that are unfamiliar with version management, here follows a brief description. You have a *working copy* or *working directory*, which is the collection of files that you work upon in your project. This is just like when you do not use version management. Additionally you have a *repository*, which is a kind of database containing the history of your project. It will contain the state of your *working copy* at certain moments in the past, together with information about who made the changes, and a description of what has changed.

If, at a certain moment, you have a state of your project that you want to keep, you *commit*, which means a copy is stored in the *repository*, together with a description that you enter. The opposite operation (i.e. making a copy from your repository to your working directory) is called *checkout*. You usually have a separate repository for each project. The repository can be on your local computer, or on a server. In the latter case it is possible that different people working on the same project use the same repository. They would then each have their own *working copy*. As they are working independently, these could be different. A version management system usually has provisions to resolve conflicting working copies.

Although these systems can store any type of file, they work best with plain text files. As our TEX sources are plain text, they are ideal candidates for using a version management system.

There are several version management systems available. One older, well-known system is *subversion* (SVN[3]). It usually has the repositories on a central server, but you can also have the repository on your local computer, if you are working alone. As SVN has only one repository per project it is called a centralised version management system.

Centralised version management systems have some big disadvantages for cooperation in teams:

- If you work together the repository must be on a central server, which means you cannot use it when you are offline.
- If you want to keep your changes registered often in the repository, then this can be confusing for the other team members. On the other hand, if you want to keep the repository relatively clean, that is, only commit major updates, then you lose the possibility to keep your own history detailed.

One solution is to have both a central repository for the team, and a local repository for your own work, but then synchronising these repositories could become tedious. However, this is where *distributed version management systems* have their strength.

In a *distributed version management system* you can have both a local repository on your computer and a central repository on a server. Or even more than one of each. Furthermore, these can be easily synchronised. The usual way to work in a team is to have a central repository for the team, and a local repository on each team member's computer. Each team member keeps a history in the local repository. This can be done often, and also offline. When changes are good enough to be put in the central repository, a team member *pushes* the local changes to the central repository, often after making one set of changes that do not reflect all the details of the work done locally. Another team member can then *fetch* these changes from the central repository when they want to be up to date. It is then probable that the newly-fetched changes are not consistent with other changes that they have made themselves in the meantime. The two sets of changes must then be *merged*. This is the basic scenario. Much more complicated workflows are also possible.

Both centralised and distributed version management systems support the concept of *branches*. A *branch* is a separate line of development in your project. For example you have a project that you publicly release from time to time. The development of this release version would for example take place on the main branch in your repository. Now after a release you want to start working on some very new experimental features for a future release. If you just continue your development, then when a bug in your release is detected, your project would be in an unstable state. So you cannot just apply a bug-fix

[3] `http://subversion.apache.org`

to the current state of your project, but you would have to go back to the state just after the release. As the repository has kept the history of your project, this is easy, but you want also to keep the current state, so that you can go back there after making the bug-fix.

Here branches come to the rescue. After your release you create a new branch for your experimental work, and continue working there. When you want to make the bug-fix, you switch back to the main branch. The repository will remember your experimental branch, and after releasing the bug-fix you can switch back to the experimental branch. If you wish you can then also *merge* the fix in your experimental branch. Later when your experiment is successful and you want to release it, you can merge it back to the main branch. You can have as many branches as you want. For example if your bug-fix is expected to be complicated, you can first try it out on a separate branch.

A very popular site for central repositories is Github.[4] This site is based on the distributed version management system Git. Git is probably the most popular version management system in use today. For my own projects I use Git exclusively nowadays, often only locally, but sometimes in combination with Github.

4.1 Use with Overleaf

To come back to the TEX project I am currently describing, it appeared that Overleaf also had Git capabilities. Although these were in beta phase at that moment, it could be used for my project. Nowadays you need a paid account on Overleaf to use the Git facilities, but because I had started using them during the beta testing, I have access to them in my free account.

Git can be used in two ways on Overleaf.

- your Overleaf project can function as a Git repository;
- your Overleaf project can be synchronised with a Github repository.

I decided to take the Github route, mainly because I have experience with Github and I could not get the direct Git repository on Overleaf working from the iPad. At this moment it is working, but its functionality is very limited compared to Github.

In order to use Git on the iPad you need a Git app. I found Git2Go,[5] which is said to be the first app to use Git on iOS. It worked well for my needs, but later I tried two others that I found: Working Copy[6] and TIG.[7] In the appendix I give a comparison of these apps.

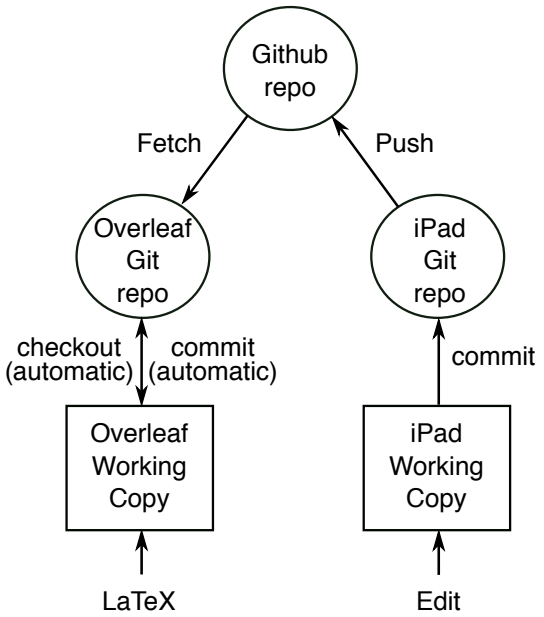


Figure 9: Git workflow

My workflow can be seen in figure 9. My editing took place in the lower right corner, on the working copy (managed by Git2Go). I could have used the editor that Git2Go provides, but it is not very sophisticated. It does not have syntax highlighting for LATEX files, and it gives no editing support beyond the standard iPad keyboard. I also had a much better text editing program called Textastic.[8] It has syntax highlighting for LATEX, good search facilities and an extended keyboard (see figure 10) that makes it easier to enter non-alphanumeric symbols. Also it has a special provision for easy cursor movement. Git2Go, and the other Git apps mentioned above, function as a kind of file system, which means that Textastic can directly edit their files without copying between the two apps. So the only extra operation to edit in Textastic rather than in Git2Go itself is switching between the apps. This extra effort I deemed worthwhile for the added comfort of using a good text editor.

After editing the file(s), I switch to Git2Go, commit the change, and immediately push it to the Github repository. Then I switch to Overleaf in the browser, fetch the changes from Github to Overleaf in the Overleaf synchronisation menu, and process the files, hopefully producing a new PDF file. Many

[4] https://www.github.com
[5] https://git2go.com
[6] https://workingcopyapp.com
[7] https://itunes.apple.com/us/app/tig-git-client/id1161732225
[8] https://www.textasticapp.com

Figure 10: Textastic extended keyboard

times it did not yet work correctly, so I had to go back to Textastic and start a new cycle.

The problem wasn't so much in the LaTeX code, as the changes there were very simple. The main problem was getting the `latexmkrc` file correct. One difficulty was that Overleaf did not have good documentation about the context in which the Latexmk program was running. Also, running it on their server did not give as much feedback as running on your own computer. Several times I had to write extra information to a text file, and then download that to the iPad to see what happened. For example, I had to make directory listings, and write them to a text file, just to see what files were generated and what their names were. And the process was a bit tedious because I had to synchronise the files as described above before each try. But after some 50 tries, everything worked perfectly. I will spare you all the attempts that I made, but in the next section I will give you the resulting `latexmkrc` file, and explain what it does.

5 Latexmk

Latexmk is a program (a Perl script) to process a LaTeX file with all the necessary `bibtex`, `makeindex` and similar calls. It will run LaTeX and these other programs as many times as is necessary to get a completely processed and stable output.

For the run-of-the-mill LaTeX file, Latexmk has enough knowledge to know what to do. However, when there are additional requirements, like a non-standard index, glossaries, etc., you must give Latexmk a recipe of how to process the various stages. The recipe is given in the `latexmkrc` file, which in fact is also a Perl script. Latexmk has an enormous number of possibilities, and its manual[9] contains 48 pages. So it took some time to get everything right.

Overleaf provides a standard `latexmkrc` file for its jobs, but as we have seen above, this is not adequate for processing the `.ins` and `.dtx` files. To make Overleaf happy, we must provide a main `.tex` file, but with our `latexmkrc` file we don't use it, so its content is unimportant.

In figure 11 the resulting `latexmkrc` for this process is given, annotated with line numbers. In the remainder of this section I explain what it does.

line 1. This sets the timezone to your local time. This is so that messages with date and time will get your local time, and not the time of Overleaf's servers, which would be useless in most cases. As I was in Bolivia at the time, the timezone was 'America/La Paz'. Now at home it would be 'Europe/Amsterdam'.

line 3-6. In a `.dtx` file the extension `.glo`, which is normally used for glossaries, is used for the list of changes. And the sorted version, to be created by `makeindex`, will be `.gls`. These lines give a recipe how to create the `.gls` file from the `.glo` file using `makeindex`.

line 8. For processing the normal index in a `.dtx` file `makeindex` needs the additional argument `-s gind.ist`.

line 10. This defines which extra file extensions we need in the process. Besides the already mentioned `.glo` and `.gls`, there is also `.glg` which is the log output of the `makeindex` command from line 5. And the `.txt` extension is used for debugging.

line 12. Here comes the trick to let Overleaf do our work. Normally it will run `pdflatex` on the main TeX file in the project, which in our case is `multirow.tex`. But you can define the `$pdflatex` variable to let it use another command. In our case we let it run the internal function `mylatex` that follows. In this function we do all the preparatory work before we run the actual `pdflatex` command.

[9] `http://mirrors.ctan.org/support/latexmk/latexmk.pdf`

Latexmkrc file:

```
1  $ENV{'TZ'} = 'America/La Paz';
2
3  add_cus_dep('glo', 'gls', 0, 'makeglo2gls');
4  sub makeglo2gls {
5      system("makeindex -s gglo.ist -o \"$_[0].gls\" \"$_[0].glo\"");
6  }
7
8  $makeindex = 'makeindex -s gind.ist -o %D %S';
9
10 push @generated_exts, 'glo', 'gls', 'glg', 'sty', 'txt';
11
12 $pdflatex = 'internal mylatex';
13 sub mylatex {
14     my @args = @_;
15     (my $base = $$Psource) =~ s/\.[^.]+$//;
16     system("tex $base.ins");
17     # backslashes are interpreted by (1) perl string (2) shell (3) sed regexp
18     # therefore we need 8 backslashes to match a single one
19     system("sed -e s/\\\\\\\\\jobname/$base/g $base.dtx > $base.tex");
20     return system("pdflatex @args");
21 }
```

Figure 11: The final latexmkrc file. The line numbers are not part of the file.

line 14. Pick up the arguments from the call to `mylatex` in the variable `@args`. This is standard Perl prose.

line 15. Latexmk puts the name of the main TEX file in `$$Psource` (see page 45 in the Latexmk manual). This line is actually a shorthand for two statements:

```
my $base = $$Psource;
$base =~ s/\.[^.]+$//;
```

The first line copies `$$Psource` to a local variable `$base`. The second line strips off anything after (and including) the last dot. So the string '`multirow.tex`' will be transformed to just '`multirow`'. I use `$$Psource` rather than just using `multirow` so that now the `latexmkrc` file is also usable for other `.dtx` files.

line 16. First we run `tex` on our `.ins` file, which would be `multirow.ins` in our case. This generates the required `.sty` files. This is to ensure that we use the new versions of our `.sty` files, rather than an outdated version in Overleaf's TEX system.

line 19. From our `.dtx` file we generate a `.tex` file where the text `\jobname` is replaced by the actual base name of our file (in our case `multirow`). This is necessary as Overleaf defines a `\jobname` of `output`. So in this case we generate `multirow.tex` from `multirow.dtx`. This `.tex` file will input `multirow.dtx` during its processing.

We do the replacement by calling the Unix program `sed`. The `\jobname` is inside a regular expression in `sed`, therefore the backslash must be doubled. But then, this command is processed by the Unix shell, which also interprets backslashes. Therefore we must double all the backslashes again. And then this command is inside a Perl string where backslashes are also interpreted. So we must double them again, and we end up with 8 backslashes to represent a single one.

line 20. Finally we run the real `pdflatex` command with the original arguments. Note that we process the new `multirow.tex` file, because that is what Overleaf expects to do. Also, because this is run in a *sandbox* (i.e. on a copy of the original files in a separate directory), this does not affect our original file.

Finally, we also show a modified `latexmkrc` file with debugging statements included in figure 12, and the corresponding output in figure 13. You see the values of `$$Psource` and `$base`, the arguments to the `pdflatex` call, and the directory listing at the end of the process. Note that in the directory listing there is a file `multirow.log`; this is the result of the call `tex multirow.ins`. Note also the generated

Latexmkrc with debugging:

```
$ENV{'TZ'} = 'America/La Paz';

add_cus_dep('glo', 'gls', 0, 'makeglo2gls');
sub makeglo2gls {
    system("makeindex -s gglo.ist -o \"$_[0].gls\" \"$_[0].glo\"");
}
$makeindex = 'makeindex -s gind.ist -o %D %S';

push @generated_exts, 'glo', 'gls', 'glg', 'sty', 'txt';

$pdflatex = 'internal mylatex';
sub mylatex {
    my @args = @_;
    Run_subst("echo \"%%B=%B %%R=%R %%S=%S %%T=%T\" > debugout.txt"); ## DEBUG ##
    system("echo '\@args' = \"@args\"               >> debugout.txt"); ## DEBUG ##
    system("echo '\$\$Psource' = \"$$Psource\"     >> debugout.txt"); ## DEBUG ##
    (my $base = $$Psource) =~ s/\.[^.]+$//;
    system("echo '\$base' = \"$base\"               >> debugout.txt"); ## DEBUG ##
    system("tex $base.ins");
    # backslashes are interpreted by (1) perl string (2) shell (3) sed regexp
    # therefore we need 8 backslashes to match a single one
    system("sed -e s/\\\\\\\\\\jobname/$base/g $base.dtx > $base.tex");
    $status = system("pdflatex @args");
    system("ls -l                                   >> debugout.txt"); ## DEBUG ##
    return $status;
}
```

Figure 12: `latexmkrc` file with debug statements

Debug output:

```
%B=output %R=output %S=multirow.tex %T=multirow.tex
@args = -synctex=1 -interaction=batchmode -recorder
        -output-directory=/compile --jobname=output
        multirow.tex
$$Psource = multirow.tex
$base = multirow
total 1176
-rw-r--r-- 1 tex tex   3871 Mar  4 14:12 README
-rw-r--r-- 1 tex tex     49 Mar  4 14:12 README.md
-rw-r--r-- 1 tex tex   1417 Mar  4 14:12 bigdelim.sty
-rw-r--r-- 1 tex tex   1234 Mar  4 14:12 bigstrut.sty
-rw-r--r-- 1 tex tex    203 Mar  4 14:12 debugout.txt
-rw-r--r-- 1 tex tex   1054 Mar  4 14:12 latexmkrc
-rw-r--r-- 1 tex tex  80398 Mar  4 14:12 multirow.dtx
-rw-r--r-- 1 tex tex   2182 Mar  4 14:12 multirow.ins
-rw-r--r-- 1 tex tex   3719 Mar  4 14:12 multirow.log
-rw-r--r-- 1 tex tex   5022 Mar  4 14:12 multirow.sty
-rw-r--r-- 1 tex tex  80398 Mar  4 14:12 multirow.tex
-rw-r--r-- 1 tex tex   3487 Mar  4 14:12 output.aux
-rw-r--r-- 1 tex tex      0 Mar  4 14:12 output.chktex
-rw-r--r-- 1 tex tex  25207 Mar  4 13:10 output.fdb_latexmk
-rw-r--r-- 1 tex tex  20593 Mar  4 14:12 output.fls
-rw-r--r-- 1 tex tex   3281 Mar  4 14:12 output.glo
-rw-r--r-- 1 tex tex   3578 Mar  4 08:13 output.gls
-rw-r--r-- 1 tex tex   3270 Mar  4 14:12 output.idx
-rw-r--r-- 1 tex tex    891 Mar  4 08:13 output.ilg
-rw-r--r-- 1 tex tex   2655 Mar  4 08:13 output.ind
-rw-r--r-- 1 tex tex  34086 Mar  4 14:12 output.log
-rw-r--r-- 1 tex tex 610336 Mar  4 14:12 output.pdf
-rw-r--r-- 1 tex tex 262970 Mar  4 14:12 output.synctex.gz
-rw-r--r-- 1 tex tex   1467 Mar  4 14:12 output.toc
```

Figure 13: `latexmkrc` debug output (the `@args` line has been broken into several lines for print)

`.sty` files. The files resulting from the `pdflatex` call on `multirow.tex/dtx` are all called `output.*`. So `makeindex` must also act on these files. In figure 11, line 5, this is accomplished because the file name is given as an argument to the function `makeglo2gls`. In line 8 it is accomplished because the patterns `%S` and `%D` are replaced by the *source* and `destination` of the command, respectively, i.e. `output.idx` and `output.ind`.

6 Conclusion

Although working at home on my MacBook is much more comfortable, it is possible to do some serious LaTeX work on your iPad while you are travelling. It takes some effort to find the proper way to do it, however. I hope this article helps you to get started if you need this work flow.

A Appendix — iOS Git apps compared

In this section I compare the three Git apps on iOS that I tried. I did all the production work in Git2Go, but after it was finished I also tried Working Copy and TIG.

Git2Go has a limitation that it only cannot work with Git repositories on all servers. It works with a limited number of services, namely Github, Bitbucket[10] and Gitlab[11]. Other remote repositories can be used if they offer access by the SSH protocol. SSH is one of the two main protocols used to connect to Git servers. The other is HTTPS. Overleaf only offers HTTPS, which Git2Go does not support.

To create a repository on your iPad you must *clone* (i.e. copy) an existing repository on one of the supported servers. You cannot create a local-only repository on the iPad. Once you have the repository on your iPad, you can edit the files in the repository, commit the changes, create new branches. It can fetch from and push to the remote repository, but these are not separate operations. It always does a fetch (which may be empty), followed by a push. It can also merge different branches. It is a limited set of operations compared to the full Git functionality, but it is sufficient for a normal workflow as described above. Also cooperation with other people would be possible as long as the more esoteric Git functionality is not required. Git2Go's editor has syntax highlighting for a limited number of programming languages.

Git2Go is free, as long as you only access *public* repositories (i.e. repositories that everybody can see). To access private repositories you would have to buy an upgrade.

For the push operation you will have to login, and Git2Go will remember your username and password, until you explicitly logout.

And occasionally it crashes.

Last minute note: I later tried to re-install Git2Go on my iPhone, and got the message that it was no longer available on the App Store. Also a search in the App Store did not come up with Git2Go. I have no idea if this is a permanent situation, or if it might be in the process of updating.

Working Copy is the nicest of the three apps. It has a very elaborate set of functions. It can connect to all kinds of servers, including Overleaf. However, to use the `push` functionality you have to pay. The price is quite steep (€17.99 at the time of writing), but you can get a free 10 day trial. I used this for writing this article to see how it worked.

Working Copy can clone from existing repositories, including through SSH and HTTPS, and also create local repositories. It can also create a local repository from a `.zip` file. Once you have a repository it can connect your repository to more than one remote repository, which sometimes can be quite handy. For example in the current example, the repository on the iPad could have been connected both to the Overleaf repository and to the Github repository. Of course you will have to be careful not to mess up your workflow.

If your iPad is connected to a Mac or PC with iTunes, you can drag and drop a repository on your computer through iTunes, and it will be copied to the iPad.

Working Copy's editor has syntax highlighting for more than 50 different languages. It can show nice graphical representations of your branches and your commit history (see figure 14). Besides the `merge` functionality it also has the `rebase` functionality, which is an alternative for merge. For cooperating in large projects this functionality is sometimes necessary.

There is more than fits in this limited space, but Working Copy is by far the best of the three apps. It is expensive, but if you do a lot of work with Git on your iPad, it is worth the price. Working Copy operates in a small market, so the price is understandable.

TIG is the third app I tried. It takes more or less a middle ground between Git2Go and Working Copy. Like Working Copy it can connect to all kinds of repositories, including Overleaf, and it has *push* functionality. And it is free. It can clone existing repositories, and create local ones. It can also connect repositories to more than one remote repository.

Its editor has syntax highlighting support for 166 languages.

However, although the functionality is great for a free app, I found its user interface sometimes confusing. And to fetch/push to your remote repositories you have to enter your username and password every time. I did not find a way in which it could remember these. This is very annoying. And it crashed quite often.

As I mentioned above, all these apps have the facility that you can open their files in an external editor. Figure 15 shows how to open files from Git2Go in Textastic. This is done inside Textastic with the

[10] `https://bitbucket.org`
[11] `https://gitlab.com`

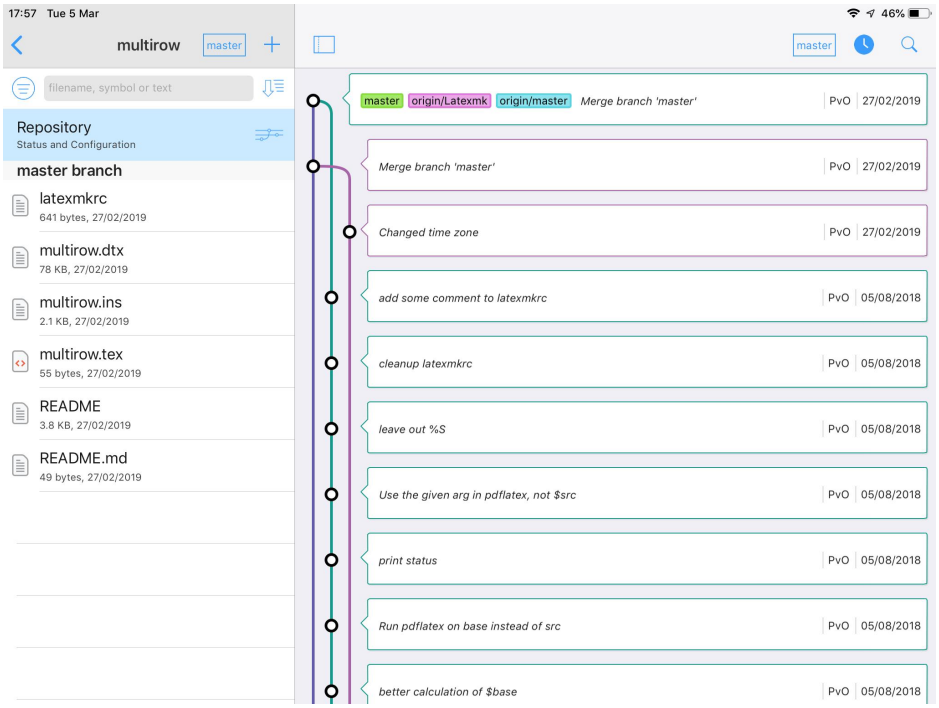


Figure 14: Graphical commit history in Working Copy

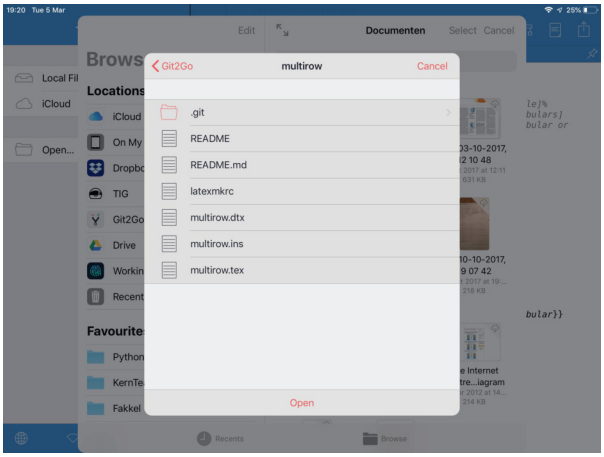


Figure 15: Opening a Git2Go file or repository in Textastic

"Open. . . " button, then selecting "Git2Go". It is then possible to choose "Open" down in the pop-up, which will open the whole directory in Textastic, or select one filename, which will open that file.

Summary:

- If you only need access to repositories hosted by Github, Bitbucket or Gitlab, or repositories that can be accessed by the SSH protocol, and your requirements are modest, you can choose Git2Go (if still available).
- If you need access to repositories that do not fall in the previous categories (such as Overleaf), and you can live with a not so optimal user interface, and your requirements are modest, you can choose TIG. It may be a good choice when you want to connect to a repository that Git2Go does not support, and when you find Working Copy too expensive.
- If you want the top Git app on your iPad (or iPhone) and are willing to pay the price, I would recommend Working Copy. If you want to do serious work with Git, this is the choice and it would be worth the price.

There are nowadays some other Git apps available, but it seems that they are roughly comparable to one of the above. Some of them only support just Github, Bitbucket or Gitlab. I have not found any free app that comes with the functionality of Git2Go or TIG. Other paid apps may be slightly cheaper than Working Copy, but they also have less functionality.

◇ Piet van Oostrum
http://piet.vanoostrum.org
piet (at) vanoostrum dot org

A Brazilian Portuguese work on MetaPost, and how mathematics is embedded in it

Estevão Vinícius Candia

Abstract

This article presents a summary of my final work [1] for the master's degree course in Mathematics, presented on November 1, 2018. This work shows an introduction to the basic commands of the MetaPost language. For this, some mathematical concepts are explained, as they are part of its programming. It is shown how these concepts can be used to create high-quality figures used in areas of mathematics such as geometry and the study of graphs of real functions.

Introduction

Written communication is the basis for teaching and learning in various areas of knowledge. Writing a legible and elegant mathematical work was not so simple in the old days without the computational tools available today. The ability to expose knowledge in accordance with technical requirements, or even personal norms, moved people to create typographic writing systems that included mathematical symbols and concepts. MetaPost is a very useful tool for this.

When I needed a research topic for my master's thesis, my advisor, Professor Elisabete Freitas, introduced me to MetaPost and said that it would be very nice to study some of the mathematics behind it. She said she uses MetaPost whenever she needs to create pictures for her work on TEX and that it has powerful tools that can contribute to the work of Mathematics teachers. So, I met this language and started studying its pedagogical use.

What did I study?

Chapter 1 of the dissertation presents some commands for creating basic figures for an interested reader who has never heard of MetaPost. To identify the positions in a drawing, MetaPost uses the one-to-one correspondence between the points of a plane and the set of ordered pairs of real numbers. Mathematics!

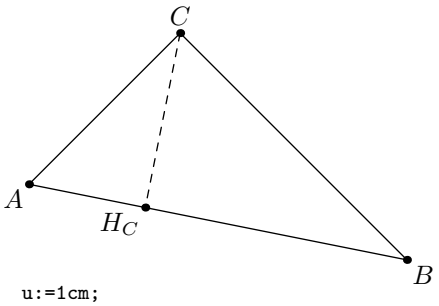
In this initial chapter, the reader learns about the commands `draw`, `drawdot`, `cycle`, paths with `--` and `..`, style commands that change the width, size, or shape of a solid or dashed line, `label`, `dotlabel`, `drawarrow`, `drawdblarrow`, some data types, commands for colors, filling and mathematical operators. It also contains the conversion table of the units of measurement used in MetaPost, as shown in Table 1 and a short tutorial on how to insert a MetaPost figure into a TEX document.

Unit of measurement	Conversion
`pt`	0,035145 cm
`bp`	0,035278 cm
`in`	2,54 cm
`pc`	0,423333 cm
`mm`	0,1 cm
`cc`	0,451167 cm
`dd`	0,0376 cm

Table 1: Conversion table of the units of measurement used in MetaPost

Some mathematical concepts of vectors and geometry can be used to make drawings in MetaPost. MetaPost has vector operators and is able to understand the data of the type pair as vectors. Chapter 2 addresses the mathematical basis for this, as well as some applications of these operations in geometry.

This chapter defines vectors and shows some vector properties. It presents the concept of colinearity and dot product (inner product) of two vectors. An application of these concepts is used to show how to plot the altitude of a triangle with the `dotprod` operator, as seen in fig. 1.




```
u:=1cm;
z1=origin; z2=(5u,-u); z3=(2u,2u);
draw z1--z2--z3--cycle;
dotlabel.llft(btex $A$ etex, z1);
dotlabel.lrt(btex $B$ etex, z2);
dotlabel.top(btex $C$ etex, z3);
(z2-z1) dotprod (z4-z3)=0;
z4=whatever[z1,z2];
draw z3--z4 dashed evenly;
dotlabel.llft(btex $H_C$ etex, z4);
```

Figure 1: Triangle ABC with relative altitude drawn on side AB

The second chapter also shows the mathematics behind the drawing of a regular polygon and the four triangle centers: centroid, circumcenter, incenter and orthocenter. In addition, it shows how to draw them with MetaPost (figs. 2–6).

Chapter 3 discusses transformations in the plane, as well as features that allow you to use them in MetaPost to draw various graphics. Some applications

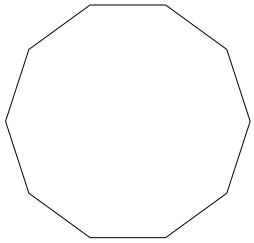


Figure 2: Decagon of side 1 cm.

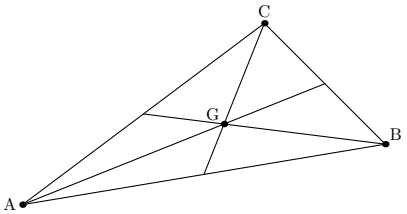


Figure 3: Medians and the centroid G of ABC.

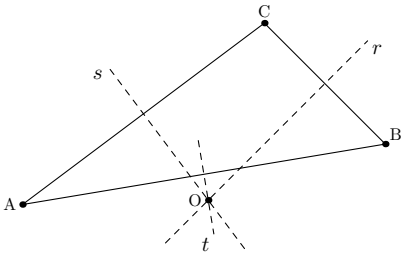


Figure 4: Perpendicular bisectors and the circumcenter O of ABC.

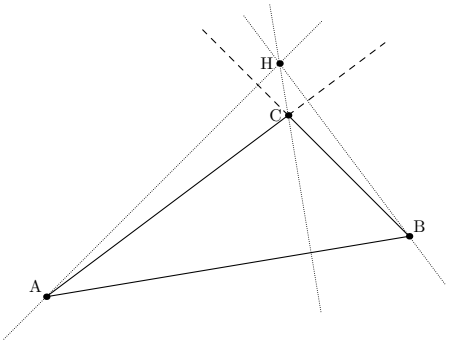


Figure 5: Altitudes and the orthocenter H of ABC.

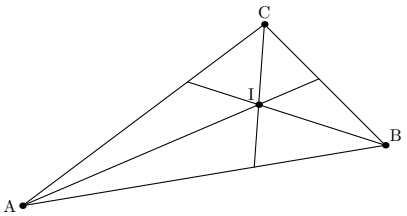


Figure 6: Angle bisectors and the incenter I of ABC.

of these transformations are also presented for the sketch of circles and ellipses.

A transformation in the plane $\mathbb{R}^2$ is a function $T : \mathbb{R}^2 \to \mathbb{R}^2$, i.e, a correspondence that associates with each point P of the plane another point $P_1 = T(P)$ of the plane, called its image by T. In MetaPost, transformations are performed using the `transform` type. With them, along with the `fullcircle` command, you can draw circles and ellipses, as shown in figures 7 and 8.

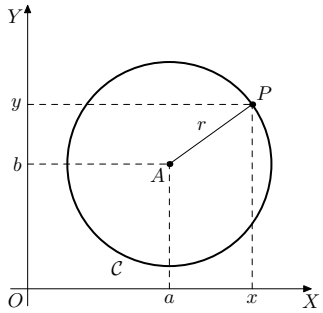


Figure 7: Circle $\mathcal{C}$ of center A and radius r

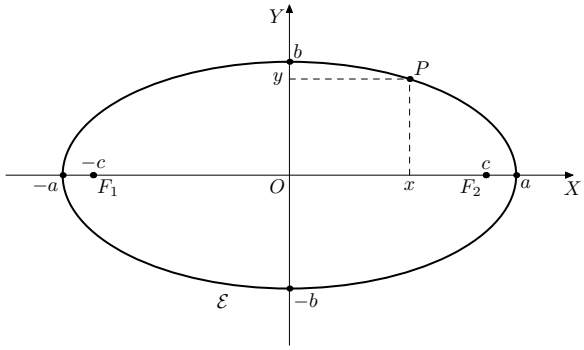
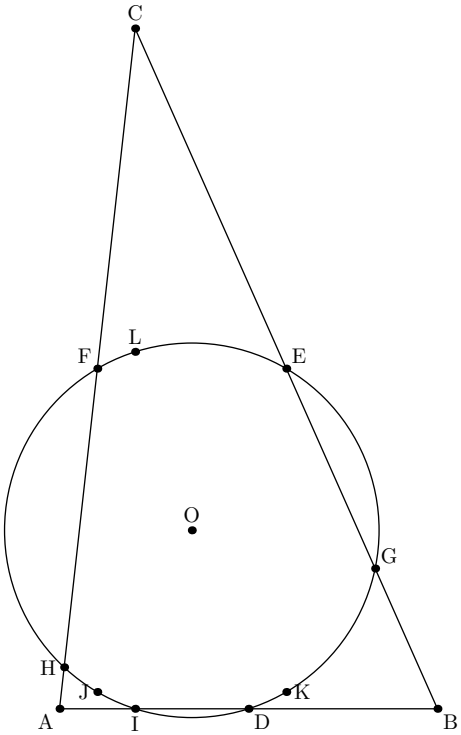


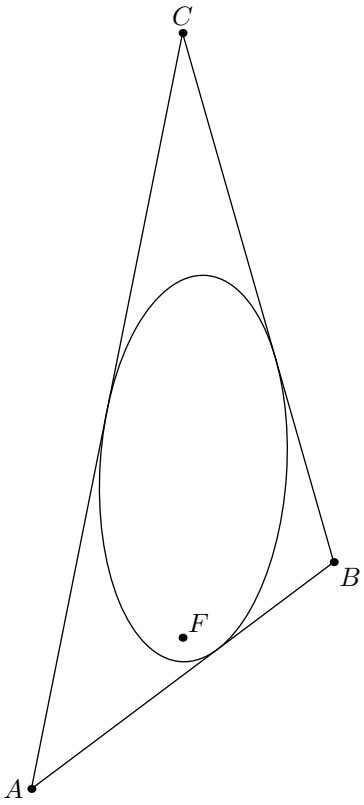
Figure 8: Ellipse $\mathcal{E}$ of center O, focuses F_1 and F_2 and major axis $2a$

Although it may seem obvious that using the `scaled`, `xscaled`, `yscaled`, and `shifted` transforms generates the circles and ellipses from the initial circle `fullcircle`, the mathematics behind this concept is accurate and, if I may say, particularly beautiful. These results are exemplified with two interesting problems of geometry: the nine-point circle and the problem of tracing the ellipse tangent to the triangle with one of its focuses an inner point of the triangle, as shown in figures 9 and 10.




```
numeric u; u=1cm;
pair A,B,C,D,E,F,G,H,I,J,K,L,O,X;
A=(0,u);
B=(5u,u);
C=(u,10u);
D=.5[A,B];
E=.5[B,C];
F=.5[A,C];
G=whatever[C,B];
H=whatever[A,C];
I=whatever[B,A];
(A-G) dotprod (B-C) =
(B-H) dotprod (C-A) =
(C-I) dotprod (A-B) = 0;
X = whatever[A,G] = whatever[B,H];
J=.5[A,X]; K=.5[B,X]; L=.5[C,X];
(O-0.5(D+I)) dotprod (D-I) =
(O-0.5(G+K)) dotprod (G-K) = 0;
draw A--B--C--cycle;
draw fullcircle
scaled (2*abs(O-D)) shifted O;
```

Figure 9: Nine-point circle




```
numeric u,a,b,c,theta;  transform T;
pair A,B,C,F[],G[],O,P[]; path E;
u=1cm; A=(-u,-3u); B=(3u,0); C=(u,7u);
draw A--B--C--cycle; F1=(u,-u); F3=whatever[A,C];
F4=whatever[A,B]; F5=whatever[C,B];
(F3-F1) dotprod (A-C)= (F4-F1) dotprod (A-B)=
(F5-F1) dotprod (C-B)=0;
G1 = F1 rotatedaround (F3,180);
G2 = F1 rotatedaround (F4,180);
G3 = F1 rotatedaround (F5,180);
(F2-0.5(G2+G3)) dotprod (G2-G3) =
(F2-0.5(G1+G2)) dotprod (G1-G2) = 0;
P1=whatever[A,C]=whatever[F2,G1]; O=0.5[F1,F2];
2a=abs(F1-P1)+ abs(F2--P1); c=abs(F1--O); b=a+-+c;
theta=angle(F2-F1); T:= identity xscaled 2a
yscaled 2b rotated theta shifted O;
E:= fullcircle transformed T; draw E;
```

Figure 10: Ellipse inscribed in the triangle ABC with one focus at F

Chapter 4 deals with the elaboration of graphs of continuous functions defined in intervals of the line. Initially a summary is made with some propositions about continuous functions. Next, some MetaPost techniques are outlined to sketch function graphs through macros (fig. 11). The techniques in this chapter discuss how to use a loop to calculate coordinates of graph points, how to cut a piece of the graph you want to see, how to use or create macros for non-predefined math functions in MetaPost, and how to plot graphs with asymptotes.

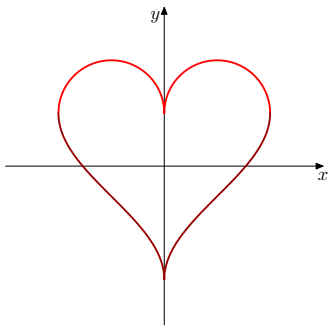
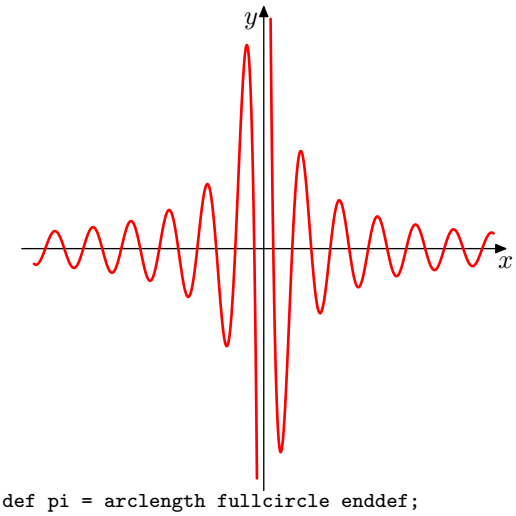


Figure 11: Functions $f(x) = 1 + \sqrt{1 - |x-1|^2}$ and $g(x) = 1 + \arccos(1 - |x|) - \pi$ (grayscaled for print)

For instance, the graph of figure 12 was sketched using some of these techniques. First, some macros are defined for the calculation of the cosine function. Then the limits of the axes of the Cartesian plane and the limits of the graph of the function are defined. Since the function is defined in two disjoint intervals, a loop is used to calculate 1000 points of each interval. Then another loop is used to define the two smooth curves through these points. The last step is to draw these two curves respecting the preset limits.




```
def pi = arclength fullcircle enddef;
numeric radian; radian := 180/pi;
vardef cos primary x = (cosd(x*radian)) enddef;
numeric u, minx, maxx, miny, maxy; path q;
u:=0.8cm; minx:=-4; maxx:=4; miny:=-4; maxy:=4;
drawarrow ((minx,0)--(maxx,0)) scaled u;
drawarrow ((0,miny)--(0,maxy)) scaled u;
label.bot (btex $x$ etex, (maxx*u,0));
label.llft (btex $y$ etex, (0,maxy*u));
q:=((minx,miny)--(maxx,miny)--(maxx,maxy)--
(minx,maxy)--cycle) scaled 0.95;
numeric h,n; path p[];
n:=1000; a:=0.1; b:=maxx; h:=(b-a)/n;
for i=0 upto n: x[i]:=a+i*h;
y[i]:=(cos (10*x[i]))/x[i]; endfor;
p[1]=z[0] for j=1 upto n: ..z[j] endfor;
a:=minx; b:=-0.1; h:=(b-a)/n;
for i=0 upto n: x[i]:=a+i*h;
y[i]:=(cos (10*x[i]))/x[i]; endfor;
p[2]=z[0] for j=1 upto n: ..z[j] endfor;
pickup pencircle scaled 1bp;
draw (p1 cutafter q cutbefore q)
scaled u withcolor red;
draw (p2 cutafter q cutbefore q)
scaled u withcolor red;
```

Figure 12: Function $f(x) = \dfrac{\cos 10x}{x}$ (grayscaled for print)

Final considerations

MetaPost is a powerful tool for using mathematical knowledge in the preparation of figures. The images generated through this language have a satisfactory professional level for what is expected of an academic work. People who teach and research mathematics can improve the presentation of their work and guide their students in the application of acquired mathematical knowledge. Thus, I hope that this work may, in a way, have contributed to the development of mathematics teachers and, consequently, to the development of their students.

I am part of a research group of the Institute of Mathematics of the Federal University of Mato Grosso do Sul, called "Fundamentals of Mathematics for High School and the use of the language MetaPost". This project proposes to provide material for the high school curriculum using TEX and MetaPost. We also intend to disseminate this language in the academic environment of Brazil, since there is very little work on MetaPost in Portuguese. If you would like to collaborate, please contact us via my email address.

References

[1] CANDIA, Estevão V. *Matemática e o MetaPost*. Master's degree thesis. UFMS, 2018. Available at `http://sca.profmat-sbm.org.br/sca_v2/get_tcc3.php?id=160350039`.

◇ Estevão Vinícius Candia
`estevao.candia (at) ifms dot edu dot br`

LaTeX News

Issue 30, October 2019

Contents

LaTeX-dev formats now available

We know that many of you, especially developers and maintainers of important packages, have a strong interest in a stable LaTeX environment.

In order to keep LaTeX very stable for users whilst allowing for further development to continue, we now have a development branch of LaTeX on GitHub containing development code for the upcoming release. When this code is ready for wider consumption and testing, we generate a pre-release of LaTeX from this development branch and make it available on CTAN.

For users of the TeX Live and MiKTeX distributions it is therefore now straightforward to test their documents and code against the upcoming LaTeX release with ease, simply by selecting a different program name (when using the command line) or by selecting a menu entry (after setting it up; see below).

If you do this then the latest version of the LaTeX development format will be used to process your document, allowing you to test the upcoming release with your own documents and packages. For example, if you run

```
pdflatex-dev myfile
```

then you will be greeted on the screen with something like `LaTeX2e <2019-10-01> pre-release-2` (identifying the pre-release format) instead of the normal `LaTeX2e <2018-12-01>`. In this pre-release you will find the latest new features that we have developed.

Our hopes

We don't expect everybody to start using the development formats to participate in testing, but we hope that people with a strong interest in a stable LaTeX environment (especially developers and maintainers of important packages) will use the new facilities and help us to ensure that future public releases of LaTeX do not (as has happened in the past) require some immediate patches because of issues that were not identified by our internal regression test suite or by other testing we do.

Any issue identified when using the development format should preferably be logged as an issue on GitHub, following the procedure outlined on our website at `https://www.latex-project.org/bugs/` including the use of the latexbug package as described.

Our bug reporting process normally states that issues involving third-party software are out of scope as we can't correct external packages; see [1]. However, in the particular case of the development format showing an incompatibility with a third-party package, it is fine to open an issue with us (in addition, please, to informing the maintainer of that package) so that we know about the problem and can jointly work on resolving it.

Details please . . .
More details and some background information about the concepts and the process are available in an upcoming *TUGboat* article: "The LaTeX release workflow and the LaTeX dev formats" [2].

Setting up menu items
While the command line call works out of the box if you have a recent TeX Live or MiKTeX installation, its use within an integrated editing environment doesn't at this point in time (maybe the developers of these editors will include it in the future). However, it is normally fairly simple to enable it as most (or even all?) of them provide simple ways to call your own setup. How this works in detail depends very much on the environment you use, so we can't give much help here.

But as an example: to provide an additional menu entry for `XeLaTeX-dev` on a MacBook all that is necessary is to copy the file `XeLaTeX.engine` to `XeLaTeX-dev.engine` and change the call from `xelatex` to `xelatex-dev` inside.

Improving Unicode handling in pdfTeX

Perhaps the most important improvement in this release is even better support for UTF-8 characters when using pdfTeX.[1]

When using a "Unicode engine", any Unicode character (that is not acting as a command, i.e., is not "active") can be used as part of the `\label`/`\ref` mechanism or can be displayed in a message or written to a file. In 8-bit engines, however, this was severely restricted: essentially you had to limit yourself to using ASCII letters, digits and a few punctuation symbols. With the new release, most of these restrictions have been removed and you now can write labels such as

```
\label{eq:größer}
```

or use accented characters, etc., as part of a `\typeout` message. The only requirement remaining is that only those UTF-8 characters that are also available for typesetting can be used, i.e., only those characters for which adequate font support is loaded. Otherwise you will get an error message stating that the particular Unicode character is not set up for use with LaTeX.

Note, however, that the restrictions on what characters can be used in the names of commands have not changed.

What is not possible when using an 8-bit engine such as pdfTeX is to use characters other than ASCII letters as part of a command name. This is due to the fact that all other characters in such engines are not single character tokens, but in fact consist of a sequence of bytes and this is not supported in command names.

[1] The Japanese engines e-pTeX and e-upTeX can't use these features yet as they don't support the primitive `\ifincsname`. Work is under way to resolve this in the engines.

Improving file name handling in pdfTeX

A related change is that file names used as part of `\input`, `\includegraphics`, etc., commands can now contain any Unicode characters allowed by the file system in use, including spaces. In this case, even characters that can't be typeset (due to lack of font support) can be used.

Improving the `filecontents` environment

The `filecontents` environment now supports an optional argument in which you can specify that it is allowed to `overwrite` an already existing file; by default nothing is written if a file with the given name exists anywhere in the search tree. An alternative name for this option is `force`. Even then the environment will refuse to write to `\jobname.tex` to avoid clobbering its own input file. However, if you use a different extension on your input file you could still overwrite it (there is no way to test for that).

There is also an option `nosearch`, which specifies that only the current directory is examined for an existing file, not the whole TeX inputs tree. This is useful if you want to write a local copy of a standard system file. Finally, `noheader` prevents writing a preamble to the file (this is the same as using the star form of the environment).

Another change is that this environment is now allowed anywhere in the document, which means it provides everything (and more) of what the now obsolete filecontents package provided.

Making more user commands robust

In the early days of LaTeX many commands were fragile, i.e., they needed `\protect` in front of them when used in places such as section headings and other "moving arguments", etc. In LaTeX 2ε many of these commands were made robust, but still a fairly large number remained unnecessarily fragile.

In this release of LaTeX we have now made a lot more commands robust. There is a very small collection of commands that must stay fragile because their expansion (maybe partially) at just the right time is critical. Yet others are unlikely to ever be needed in a "moving argument".

Doing this for `\begin` and `\end` was rather tricky as the standard mechanism with `\DeclareRobustCommand` doesn't work here, at least not for `\end` as that needs to expand during typesetting without generating a `\relax` (from the `\protect`). Such a token would start a new row in table environments, such as `tabular`, etc. Furthermore, some packages try to look into the definition of `\end` by expanding it several times. Thus expansion with `\expandafter` had to produce exactly the same result as before. But in the end we overcame

that hurdle too, so now environments are automatically robust if used in places like headings or `\typeout` and so forth.

What hasn't been tackled yet is the redefinitions in amsmath: this package redefines a number of basic math constructs that are now robust, so that they become fragile again once the package is loaded. This area will be addressed in a followup release. *(github issue 123)*

Other changes to the LaTeX kernel

Guard against `\unskip` in tabular cells

If a `tabular` or `array` cell started with a command that started with an `\unskip` then centering the column broke because the stretching glue on the left got removed. The fix for this was to add a minuscule, and hence unnoticeable, additional space after the stretching space: removing this extra space causes no problems.

This change was also applied in the array package. *(github issue 102)*

Fix Unicode table data

`U+012F` which is "i with ogonek" produced a "dotless i with ogonek" by mistake. This has been corrected. *(github issue 122)*

The Unicode slots `27E8` and `27E9` have been mapped to `\textlangle` and `\textrangle` which is the recommended mapping. In the past they raised a LaTeX error. *(github issue 110)*

When doing cut-and-paste from other documents or websites, f-ligatures and others ligatures might end up as single Unicode characters in your file. In the past those got rejected by LaTeX. We now define those Unicode slots and map them back to the sequence of individual characters constituting the ligature. If supported by the current font (which is normally the case) they are then reconstructed as ligatures and thus get typeset as desired. Otherwise they will come out as individual characters which is still better than an error message. *(github issue 154)*

Improve `\InputIfFileExists`'s handling of file names

In rare circumstances it was possible that `\InputIfFileExists` would work incorrectly, e.g., a construction such as

```
\InputIfFileExists{foo}{\input{bar}}{}
```

would not load the files `foo.tex` and `bar.tex` but would load `bar.tex` twice. This has been corrected. *(github issue 109)*

Improve interface for cross-references

The packages fncylab and varioref provided a slightly improved definition of `\refstepcounter` which allowed the internal `\p@..` commands to receive the counter value as an argument, instead of acting as a simple prefix. This supports more complex formatting of the value in the reference.

These packages also provided the command `\labelformat` to help in the specification of such formatting in an easy way. For example, `\labelformat{equation}{eq.~(#1)}` specifies that references to equations automatically come out as "eq. (5)" or similar. As such a `\labelformat` declaration means a `\ref` command can no longer be successfully used at the start of a sentence, the packages also provided `\Ref` for such scenarios.

Both of these commands, `\labelformat` and `\Ref`, are now removed from the packages and instead made available in the kernel so there is no need to load additional packages.

Improve wording of a warning message

The kernel now says "Trying to load ..." instead of "Try loading ..." in one of its informal messages to match style of similar messages. *(github issue 107)*

Avoid bad side-effects of `\DeclareErrorFont`

As a side effect of setting up the error font for NFSS, this declaration also changed the current font size back to 10pt. In most circumstances that doesn't matter, because that declaration was meant to be used only during the format generation and not during a LaTeX run. However, it has turned out to be used by some developers in other places (incorrectly in fact: e.g., inside some `.fd` files) where resetting the size causes havoc seemingly at random. The command has now changed to not produce such side effects. *(gnats issue latex/4399)*

nfssfont: Make font table generation the default action

With the small file `nfssfont.tex` it is possible to produce font tables and other font tests in the style set up by Don Knuth. In nearly all cases a font table is wanted, so this action has been made the default. Now one can simply hit enter instead of having to write `\table\bye`.

trace: Add package support in the kernel

The trace package implements the commands `\traceon` and `\traceoff` that work like `\tracingall` but skip certain code blocks that produce a lot of tracing output. This is useful when debugging, to suppress uninteresting tracing from, for example, loading a font. Code blocks that should not be traced need to be surrounded by the commands `\conditionally@traceoff` and `\conditionally@traceon`.

The LaTeX kernel now provides dummy definitions for these two commands so that package writers can use them in their packages regardless of trace being loaded or not.

Changes to packages in the tools category

array: Warn if primitive column specifiers are overwritten

With `\newcolumntype` it is possible to define your own column specifiers for a `tabular` preamble; it is also possible to change existing ones. However, doing that for a primitive column specifier, such as `c`, is seldom a good idea, since then its functionality becomes unavailable. The package was therefore supposed to warn the user in this case, but due to a missing `\expandafter` in the code it never did—now it does. *(github issue 148)*

multicol: Introduce `minrows` counter for balancing

When there are only a few lines of text on a page at the end of a `multicols` environment, balancing the columns often looks rather odd: such as three columns each containing a single line. The balancing behavior can now be controlled through the counter `minrows` (default is 1) which specifies that, after balancing, there must be at least that many lines in the first column. Thus, if you set `minrows` to `2` then you would get a distribution of `2+1+0` lines and if set to three, the result would be `3+0+0` instead of the default `1+1+1`.

What is most appropriate really depends on the circumstances, but this now gives you the tools to make local or global adjustments.

varioref: Better support for cleveref

The varioref package has been internally updated to provide better interfaces for packages such as hyperref and cleveref.

It also has a new package option `nospace` that stops varioref from meddling with space in front of its commands. The original behavior was always somewhat problematical and it is suggested that all new documents use this option (which should really have been the default).

Support was also added for the Arabic language through the option `arabic`.

xr: Support citations to bibliographies in external documents

The xr package can be used to cross-reference an external LaTeX document. This means that even when a work is split over different documents (that need to be processed separately), `\ref` or `\pageref` can use labels from any document, creating links between them. This facility has now been extended so that `\cite` commands and their cousins can now also reference bibliographies in external documents; this feature was first provided in the package xcite by Enrico Gregorio.

Note that for technical reasons xr doesn't work with hyperref. Use xr-hyper instead if you need the latter package.

Changes to packages in the amsmath category

amsmath: Introduce `\overunderset` command

The amsmath package has always offered the commands `\overset` and `\underset` to produce binary operators with something set above or below. But sometimes one needs to put something above and something below: The newly added `\overunderset` makes this easily possible.

Documentation updates

There are a number of documentation updates in files on the documentation page of the project website [4].

Highlighting the standard NFSS codes for series

The *Font Selection Guide* [3] has been updated to strongly recommend that the standard codes should be used when providing font support. The reason for this recommendation is explained here.

The font selection scheme uses a number of standard codes for `\fontseries` and `\fontshape` to ensure that different fonts are comparable, e.g., that you get a "light" weight if you specify `l` and "extra bold" when you write `eb`, etc. Over the years people came up with a number of other creative short codes like `k`, `j`, `t` and others with the result that changing a font family required different codes and thus prevented users from easily mixing and matching different families. Some work has been undertaken to get back to a coherent scheme and all the font families supported through the program `autoinst` are now producing the standard codes again.

LaTeX `base` and `doc` distribution reunited

For a long time the LaTeX distribution available from CTAN was split into several parts to allow them to be uploaded or downloaded separately. As this is these days more confusing than helpful we have recombined the base part with the documentation part (as both are anyway always updated together). Thus the package `latex-doc` is no longer separately available from CTAN but contained in the `latex-base` distribution.

References

[1] Frank Mittelbach: *New rules for reporting bugs in the LaTeX core software*. In: TUGboat, 39#1, 2018. `https://latex-project.org/publications/`

[2] Frank Mittelbach: *The LaTeX release workflow and the LaTeX dev formats*. In: TUGboat, 40#2, 2019. `https://latex-project.org/publications/`

[3] LaTeX Project Team: *LaTeX 2ε font selection*. `https://latex-project.org/documentation/`

[4] *LaTeX documentation on the LaTeX Project Website*. `https://latex-project.org/documentation/`

Understanding scientific documents with synthetic analysis on mathematical expressions and natural language*

Takuto Asakura

1 Introduction

Converting Science, Technology, Engineering, and Mathematics (STEM) documents to formal expressions has a large impact on academic and industrial society. It enables us to construct databases of mathematical knowledge, search for formulae, and develop a system that generates executable code.

However, such conversion is an exceedingly ambitious goal. Mathematical expressions are commonly used in scientific communication in numerous fields such as mathematics and physics, and in many cases, they express key ideas in STEM documents. Despite the importance of mathematical expressions, formulae and texts are complementary to each other, and neither can be understood independently. Thus, deep synthetic analyses on natural language and mathematical expressions are necessary.

To date, much effort has been made for developing Natural Language Processing (NLP) techniques, including semantic parsing (SP) [4], but their targets are mostly 'general' texts. Naturally, conventional NLP techniques include only limited features to treat formulae and numerous linguistic phenomena specific to STEM documents [3].

Meanwhile, semantics on mathematical expressions also has been deeply investigated. Such results can be seen in logic theories, the MathML specification [1], etc. However, there is a large distance between formal expressions such as first-order logic and actual formulae in natural language texts.

2 Research goals

There is substantial work remaining to achieve conversion from STEM documents to a computational form (Figure 1). At first, we are going to focus on the two foundational parts for the synthetic analyses. The first is token-level analyses on formulae. The main part of the analyses is associating formulae tokens to mathematical objects and text fragments (Section 2.1). This is a fundamental step for the conversion, but it is still almost untouched. The second step is the morphology of mathematical expression and semantics covering both formulae and texts (Section 2.2). Studying underlying theories is essential to deeply understand the structure of STEM documents. We aim for practical applications via a bottom-up approach.

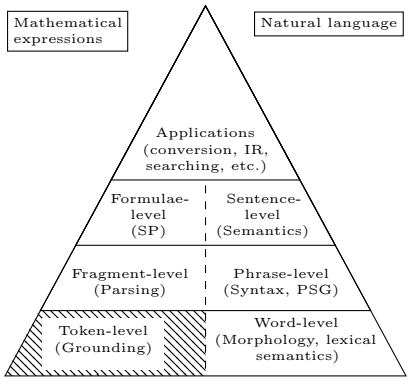


Figure 1: Overview of our task definitions. At first, we are tackling the token-level analyses on mathematical expressions (Section 2.1) and theories covering both formulae and texts (Section 2.2).

2.1 Associating tokens in formulae with mathematical objects and their descriptions in texts

Tokens in formulae (e.g., x, ε, $\times$, log) and their combinations can refer to *mathematical objects*. We human beings are able to detect what each token or combination pointing to, by using common sense, domain knowledge, and referencing descriptions in the document or in the others. This detection is fundamental and should be one of the initial steps for understanding STEM documents, but unfortunately, it cannot be easily done by a machine. There are at least four factors which make the detection highly challenging: (1) ambiguity of tokens, (2) syntactic ambiguity of formulae, (3) need for "common" sense and domain knowledge, and (4) severe abbreviation. These difficulties often appear in formulae; giving an example for (1) as a representative, already in the first chapter of a book *Pattern Recognition and Machine Learning* (PRML) [2], a character $\mathbf{y}$ (letter 'y' in bold roman) is used with several meanings including a function, vectors, and a value (Table 1).

The other part of the initial steps of understanding STEM documents is connecting text fragments to the subjective mathematical objects. Our hypothesis is that for this step, general NLP approaches such as dependency parsing are more or less applicable. Of course, some tuning for STEM documents will be required. Also, this process might need to be done by considering the result of mathematical object detection for formulae.

2.2 Semantics and morphology

Semantics on natural language and mathematical expressions have been studied separately. However, to understand STEM documents, it is important to

* A version of this extended abstract was published at the Doctoral Programme of CICM 2019.

Table 1: Usage of character $\mathbf{y}$ in the first chapter of PRML (except exercises). Underlines by the author.

Text fragment from PRML Chap. 1	Meaning of $\mathbf{y}$
... can be expressed as a function $\underline{\mathbf{y}}(\mathbf{x})$ which takes ...	a function which takes an image as input
... an output vector $\underline{\mathbf{y}}$, encoded in ...	an output vector of function $\mathbf{y}(\mathbf{x})$
... two vectors of random variables $\mathbf{x}$ and $\underline{\mathbf{y}}$ ...	a vector of random variables
Suppose we have a joint distribution $p(\mathbf{x}, \underline{\mathbf{y}})$ from ...	a part of pairs of values, corresponding to $\mathbf{x}$

investigate a synthetic semantics covering both of texts and formulae.

Though morphology has been studied for natural languages, this is not so much the case for formulae. As a matter of fact, in terms of morphology, *words* also exist in formulae. For instance, a token M is a word in "Matrix M", but M is not a word in "An entry $M_{i,j}$" ($M_{i,j}$ is a word). Unlike morphemes in natural language, tokens in formulae do not have lexical categories, but some symbols (e.g., parentheses and equal sign) and positional information (e.g., super/subscript) have typical usages.

3 Completed and remaining research

For the beginning of our research, we simplified the detection task which we described in Section 2.1. Specifically, we are making annotations on some research papers in the following manner:

1. Detecting minimal groups of tokens (we call them *chunks*), each referring to a mathematical object (chunking).
2. Categorizing chunks by the mathematical object they referring to.

This annotation (*pilot annotation*) is the fundamental process for creating the first gold dataset for associating tokens and mathematical objects. The annotated data will also be helpful for investigating the morphology on mathematical expressions.

In other words, we defined a classification task before annotating descriptions for formulae tokens. Since there are many ways to describe a mathematical object, this classification can be done more coherently through the pilot annotation. Moreover, we are expecting that the classification is naturally rather easier to be automated than giving descriptions automatically in the first attempt.

Besides the pilot annotation, all the jobs that have to be done to achieve our goal remain. For the next step, we are planning to automate the annotation process by using features such as apposition nouns and syntactic information in formulae. At the same time, we have to decide the form of mathematical objects. For now, we can say that every mathematical object should have a description and some attributes such as types (e.g., int and float). What attributes are necessary and sufficient is still not clear, and we will find out after trying the annotation for several documents.

4 Publication plans and evaluation plans

Currently, we are creating a new language resource as the pilot annotation, and we are planning to publish it for the community of language resources. For the further future, we will develop automation algorithms for mathematical object detection, which are works suitable for NLP and the digital mathematical library community, including CICM. The analyses on underlying morphology and semantics are more like works in computational linguistics.

For the initial dataset, it is better to ensure agreement among a few experts, if possible. Subsequent progress on developing algorithms and analyses on linguistic phenomena should be evaluated with our handmade gold datasets.

Acknowledgments. I would like to thank my supervisors Prof. Yusuke Miyao (the University of Tokyo) and Prof. Akiko Aizawa (National Institute of Informatics) for their most helpful advice. This work was supported by JST CREST Grant Number JPMJCR1513, Japan.

References

[1] R. Ausbrooks, S. Buswell, et al. *Mathematical Markup Language (MathML) 3.0 Specification*. World Wide Web Consortium (W3C), 2014.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[3] M. Kohlhase and M. Iancu. Co-representing structure and meaning of mathematical documents. *Sprache und Datenverarbeitung, Intl. J. for Language Data Processing* 38(2):49–80, 2014.

[4] S. Reddy, O. Täckström, et al. Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics* 4:127–140, Dec. 2016.

◇ Takuto Asakura
The Graduate University for Advanced Studies, SOKENDAI
Department of Informatics
Chiyoda, Tokyo, Japan
tkt.asakura (at) gmail dot com

Modern Type 3 fonts

Hans Hagen

Support for Type 3 fonts has been on my agenda for a couple of years now. The reason is that they might be useful for embedding (for instance) runtime graphics (such as symbols) in an efficient way. In TEX systems Type 3 fonts are normally used for bitmap fonts, the PK output that comes via METAFONT. Where for instance Type 1 fonts are defined using a set of font specific rendering operators, a Type 3 font can contain arbitrary code, in PDF files these are PDF (graphic and text) operators.

A program like LuaTEX supports embedding of several font formats natively. A quick summary of relevant formats is the following:[1]

Type 1: these are outline fonts using `cff` descriptions, a compact format for storing outlines. Normally up to 256 characters are accessible but a font can have many more (as Latin Modern and TEX Gyre demonstrate).

OpenType: these also use the `cff` format. As with Type 1 the outlines are mostly cubic Bezier curves. Because there is no bounding box data stored in the format the engine has to pseudo-render the glyphs to get that information. When embedding a subset the backend code has to flatten the subroutine calls, which is another reason the `cff` blob has to be disassembled.

TrueType: these use the `ttf` format which uses quadratic B-splines. The font can have a separate kerning table and stores information about the bounding box (which is then used by TEX to get the right heights and depths of glyphs). Of course those details never make it into the PDF file as such.

Type 3: as mentioned this format is (traditionally) used to store bitmap fonts but as we will see it can do more. It is actually the easiest format to deal with.

In LuaTEX any font can be a "wide" font, therefore in ConTEXt a Type 1 font is not treated differently than an OpenType font. The LuaTEX backend can even disguise a Type 1 font as an OpenType font. In the end, as not that much information ends up in the PDF file, the differences are not that large for the first three types. The content of a Type 3 font is less predictable but even then it can have for instance a `ToUnicode` vector so it has no real disadvantages in, say, accessibility. In ConTEXt LMTX, which uses LuaMetaTEX without any backend, all is dealt with in Lua: loading, tweaking, applying and embedding.

[1] Technically one can embed anything in the PDF file.

The difference between OpenType and TrueType is mostly in the kind of curves and specific data tables. Both formats are nowadays covered by the OpenType specification. If you Google for the difference between these formats you can easily end up with rather bad (or even nonsense) descriptions. The best references are `https://en.wikipedia.org/wiki/B%C3%A9zier_curve` and the ever-improving `https://docs.microsoft.com/en-us/typography` website.

Support for so-called variable fonts is mostly demanding of the front-end because in the backend it is just an instance of an OpenType or TrueType font being embedded. In this case the instance is generated by the ConTEXt font machinery which interprets the `cff` and `ttf` binary formats in doing so. This feature is not widely used but has been present from the moment these fonts showed up.

Type 3 fonts don't have a particularly good reputation, which is mainly due to the fact that viewers pay less attention in displaying them, at least that was the case in the past. If they describe outlines, then all is okay, apart from the fact that there is no anti-aliasing or hinting but on modern computers that is hardly an issue. For bitmaps the quality depends on the resolution and traditionally TEX bitmap fonts are generated for a specific device, but if you use a decent resolution (say 1200 dpi) then all should be okay. The main drawback is that viewers will render such a font and cache the (then available) bitmap which in some cases can have a speed penalty.

Using Type 3 fonts in a PDF backend is not something new. Already in the pdfTEX era we were playing with so-called PDF glyph containers. In practice that worked okay but not so much for MetaPost output from METAFONT fonts. As a side note: it might actually work better now that in Metafun we have some extensions for rendering the kind of paths used in fonts. But glyph containers were dropped long ago already and Type 3 was limited to traditional TEX bitmap inclusion. However, in LuaMetaTEX it is easier to mess around with fonts because we no longer need to worry about side effects of patching font related inclusion (embedding) for other macro packages. All is now under Lua control: there is no backend included and therefore no awareness of something built-in as Type 3.

So, as a prelude to the 2019 ConTEXt meeting, I picked up this thread and turned some earlier experiments into production code. Originally I meant to provide support for MetaPost graphics but that is still locked in experiments. I do have an idea for its

interface, now that we have more control over user interfaces in Metafun.

In addition to 'just graphics' there is another candidate for Type 3 fonts — extensions to OpenType fonts:

1. Color fonts where stacked glyphs are used (a nice method).
2. Fonts where SVG images are used.
3. Fonts that come with bitmap representations in PNG format.

It will be no surprise that we're talking of emoji fonts here although the second category is now also used for regular text fonts. When these fonts showed up support for them was not that hard to implement and (as often) we could make TEX be among the first to support them in print (often such fonts are meant for the web).

For category one, the stacked shapes, the approach was to define a virtual font where glyphs are flushed while backtracking over the width in order to get the overlay. Of course color directives have to be injected too. The whole lot is wrapped in a container that tells a PDF handler what character actually is represented. Due to the way virtual fonts work, every reference to a character results in the same sequence of glyph references, (negative) kern operations and color directives plus the wrapper in the page stream. This is not really an issue for emoji because these are seldom used and even then in small quantities. But it can explode a PDF page stream for a color text font. All happens at runtime and because we use virtual fonts, the commands are assembled beforehand for each glyph.

For the second category, SVG images, we used a different approach. Each symbol was converted to PDF using Inkscape and cached for later use. Instead of injecting a glyph reference, a reference to a so-called `XForm` is injected, again with a wrapper to indicate what character we deal with. Here the overhead is not that large but still present as we need the so-called 'actual text' wrapper.

The third category is done in a similar way but this time we use GraphicsMagick to convert the images beforehand. The drawbacks are the same.

In ConTEXt LMTX a different approach is followed. The PDF stream that stacks the glyphs of category one makes a perfect stream for a Type 3 character. Apart from some juggling to relate a Type 3 font to an OpenType font, the page stream just contains references to glyphs (with the proper related Unicode slot). The overhead is minimal.

For the second category ConTEXt LMTX uses its built-in SVG converter. The XML code of the shape is converted to (surprise): MetaPost. We could go directly to PDF but the MetaPost route is cheap and we can then get support for color spaces, transformations, efficient paths and high quality all for free. It also opens up the possibility for future manipulations. The Type 3 font eventually has a sequence of drawing operations, mixed with transformations and color switches, but only once. Most of the embedded code is shared with the other categories (a plug-in model is used).

The third category follows a similar route but this time we use the built-in PNG inclusion code. Just like the other categories, the page stream only contains references to glyphs.

It was interesting to find that most of the time related to the inclusion went into figuring out why viewers don't like these fonts. For instance, in Acrobat there needs to be a glyph at index zero and all viewers seem to be able to handle at most 255 additional characters in a font. But once that, and a few more tricks, had become clear, it worked out quite well. It also helps to set the font bounding box to all zero values so that no rendering optimizations kick in. Also, some dimensions can are best used consistently. With SVG there were some issues with reference points and bounding boxes but these could be dealt with. A later implementation followed a slightly different route anyway.

The implementation is reasonably efficient because most work is delayed till a glyph (shape) is actually injected (and most shapes in these fonts aren't used at all). The viewers that I have installed, Acrobat Reader, Acrobat X, and the mupdf-based SumatraPDF viewer can all handle the current implementation.

An example of a category one font is Microsoft's `seguiemj`. I have no clue about the result in the future because some of these emoji fonts change every now and then, depending also on social developments. This is a category one font which not only has emoji symbols but also normal glyphs:

```
\definefontfeature[colored][default][colr=yes]
\definefont[TestA][file:seguiemj.ttf*colored]
\definesymbol[bug1]
  [\getglyphdirect{file:seguiemj.ttf*colored}
   {\char"1F41C}]
\definesymbol[bug2]
  [\getglyphdirect{file:seguiemj.ttf*colored}
   {\char"1F41B}]
```

The example below demonstrates this by showing the graphic glyph surrounded by the `x` from the emoji font, and from a regular text font.

```
{\TestA x\char"1F41C x\char"1F41B x}\quad
{x\symbol[bug1]x\symbol[bug2]x}\quad
```

```
{\showglyphs x\symbol[bug1]x\symbol[bug2]x}%
```

In this mix we don't use a Type 3 font for the characters that don't need stacked (colorful) glyphs, which is more efficient. So the `x` characters are references to a regular (embedded) OpenType font.

The next example comes from a manual and demonstrates that we can (still) manipulate colors as we wish.

```
\definecolor[emoji-red]    [r=.4]
\definecolor[emoji-blue]   [b=.4]
\definecolor[emoji-green]  [g=.4]
\definecolor[emoji-yellow] [r=.4,g=.5]
\definecolor[emoji-gray]   [s=1,t=.5,a=1]

\definefontcolorpalette[emoji-red]
  [emoji-red,emoji-gray]

\definefontcolorpalette[emoji-green]
  [emoji-green,emoji-gray]

\definefontcolorpalette[emoji-blue]
  [emoji-blue,emoji-gray]

\definefontcolorpalette[emoji-yellow]
  [emoji-yellow,emoji-gray]

\definefontfeature[seguiemj-r][default]
     [ccmp=yes,dist=yes,colr=emoji-red]
\definefontfeature[seguiemj-g][default]
     [ccmp=yes,dist=yes,colr=emoji-green]
\definefontfeature[seguiemj-b][default]
     [ccmp=yes,dist=yes,colr=emoji-blue]
\definefontfeature[seguiemj-y][default]
     [ccmp=yes,dist=yes,colr=emoji-yellow]

\definefont[MyColoredEmojiR][seguiemj*seguiemj-r]
\definefont[MyColoredEmojiG][seguiemj*seguiemj-g]
\definefont[MyColoredEmojiB][seguiemj*seguiemj-b]
\definefont[MyColoredEmojiY][seguiemj*seguiemj-y]
```

Let's look in more detail at the woman emoji. On the left we see the default colors, and on the right we see our own:

The multi-color variant in ConTEXt MkIV ends up as follows in the page stream:

```
/Span << /ActualText <feffD83DDC69> >> BDC
q
0.000 g
BT
/F8 11.955168 Tf
1 0 0 1 0 2.51596 Tm [<045B>]TJ
0.557 0.337 0.180 rg
1 0 0 1 0 2.51596 Tm [<045C>]TJ
1.000 0.784 0.239 rg
1 0 0 1 0 2.51596 Tm [<045D>]TJ
0.000 g
1 0 0 1 0 2.51596 Tm [<045E>]TJ
0.969 0.537 0.290 rg
1 0 0 1 0 2.51596 Tm [<045F>]TJ
0.941 0.227 0.090 rg
1 0 0 1 0 2.51596 Tm [<0460>]TJ
ET
Q
EMC
```

and the reddish one becomes:

```
/Span << /ActualText <feffD83DDC69> >> BDC
q
0.400 0 0 rg 0.400 0 0 RG
BT
/F8 11.955168 Tf
1 0 0 1 0 2.51596 Tm [<045B>]TJ
1 g 1 G /Tr1 gs
1 0 0 1 0 2.51596 Tm [<045C>1373<045D>1373
                <045E>1373<045F>1373<0460>]TJ
ET
Q
EMC
```

Each time this shape is typeset these sequences are injected. In ConTEXt LMTX we get this in the page stream:

```
BT
/F2 11.955168 Tf
1 0 0 1 0 2.515956 Tm [<C8>] TJ
ET
```

This time the composed shape ends up in the Type 3 character procedure. In the colorful case (reformatted because it actually is a one-liner):

```
2812 0 d0
0.000 g              BT /V8 1 Tf [<045B>] TJ ET
0.557 0.337 0.180 rg BT /V8 1 Tf [<045C>] TJ ET
1.000 0.784 0.239 rg BT /V8 1 Tf [<045D>] TJ ET
0.000 g              BT /V8 1 Tf [<045E>] TJ ET
0.969 0.537 0.290 rg BT /V8 1 Tf [<045F>] TJ ET
0.941 0.227 0.090 rg BT /V8 1 Tf [<0460>] TJ ET
```

and in the reddish case (where we have a gray transparent color):

```
2812 0 d0
0.400 0 0 rg 0.400 0 0 RG
BT /V8 1 Tf [<045B>] TJ ET
1 g 1 G /Tr1 gs
BT /V8 1 Tf [<045C>] TJ ET
BT /V8 1 Tf [<045D>] TJ ET
BT /V8 1 Tf [<045E>] TJ ET
BT /V8 1 Tf [<045F>] TJ ET
BT /V8 1 Tf [<0460>] TJ ET
```

but this time we only get these verbose compositions once in the PDF file. We could optimize the last variant by a sequence of indices and negative kerns but it hardly pays off. There is no need for `ActualText` here because we have an entry in the `ToUnicode` vector: `<C8> <D83DDC69>`.

To be clear, the `/V8` is a sort of local reference to a font which can have an `/F8` counterpart elsewhere. These Type 3 fonts have their own resource references and I found it more clear to use a different prefix in that case. If we only use a few characters of this kind, of course the overhead of extra fonts has a penalty but as soon we refer to more characters we gain a lot.

When we have SVG fonts, the gain is a bit less. The PDF resulting from the MetaPost run can of course be large but they are included only once. In MkIV these would be (shared) so-called XForms. In the page stream we then see a simple reference to such an XForm but again wrapped into an `ActualText`. In LMTX we get just a reference to a Type 3 character plus of course an extra font.

The `emojionecolor-svginot` font is somewhat problematic (although maybe in the meantime it has become obsolete). As usual with new functionality, specifications are not always available or complete (especially when they are application specs turned into standards). This is also true with for instance SVG fonts. The current documentation on the Microsoft website is reasonable and explains how to deal with the viewport but when I first implemented support for SVG it was more trial and error. The original implementation in ConTEXt used Inkscape to generate PDF files with a tight bounding box and mix that with information from the font (in MkIV and the generic loader we still use this method). This results in a reasonable placement for emoji (that often sit on top of the baseline) but not for characters. More accurate treatment, using the origin and bounding box, fail for graphics with bad viewports and strange transform attributes. Now one can of course argue that I read the specs wrong, but inconsistencies are hard to deal with. Even worse is that successive versions of a font can demand different hacks. (I would not be surprised if some programs have built-in heuristics for some fonts that apply fixes.) Here is an example:

```
<svg transform="translate(0 -1788) scale(2.048)"
     viewBox="0 0 64 64" ...>
  <path d="... all within the viewBox ..." .../>
</svg>
```

It is no problem to scale up the image to normal dimensions, often 1000, or 2048 but I've also seen 512. The 2.048 suggests a 2048 unit approach, as does the 1788 shift. Now, we could scale up all dimensions by 1000/64 and then multiply by 2.048 and eventually shift over 1788, but why not scale the 1788 by 2.048 or scale 64 by 2.048? Even if we can read the standard to suit this specification it's just a bit too messy for my taste. In fact I tried all reasonable combinations and didn't (yet) get the right result. In that case it's easier to just discard the font. If a user complains that it (kind of) worked in the past, a counter-argument can be that other (more recent) fonts don't work otherwise. In the end we ended up with an option: when the `svg` feature value is `fixdepth` the vertical position will be fixed.

```
\definefontfeature[whatever][default]
  [color=yes,svg=fixdepth]
\definefont[TestB]
  [file:emojionecolor-svginot.ttf*whatever]
```

X  X

The newer `emojionecolor` font doesn't need this because it has a `viewBox` of `0 54.4 64 64` which fixes the baseline.

```
\definefontfeature[whatever][default]
  [color=yes,svg=yes]
\definefont[TestB]
  [file:emojionecolor.otf*whatever]
```

X  X

Another example of a pitfall is running into category one glyphs made from several pieces that all have the same color. If that color is black, one starts to wonder what is wrong. In the end the ConTEXt code was doing the right thing after all, and I would not be surprised if that glyph gets colors in an update of the font. For that reason it makes no sense to include them as examples here. After all, we can hardly complain about free fonts (and I'm also guilty of imposing bugs on users). By the way, for the font referred to here (`twemojimozilla`), another pitfall was that all shapes in my copy had a zero bounding box which means that although a width is specified, rendering in documents can give weird side effects. This can be corrected by the `dimensions` feature that forces the ascender and descender values to be used.

```
\definefontfeature[colored:x][default]
  [colr=yes]
\definefontfeature[colored:y][default]
  [colr=yes,dimensions={1,max,max}]
\definefont[TestC]
  [file:twemojimozilla.ttf*colored:x]
\definefont[TestD]
  [file:twemojimozilla.ttf*colored:y]
```

An example of a PNG-enhanced font is the `applecoloremoji` font. The bitmaps are relatively large for the quality they provide and some look like scans.

```
\definefontfeature[sbix][default][sbix=yes]
\definefont[TestE]
  [file:applecoloremoji.ttc*sbix at 10bp]
```

As mentioned above, there are fonts that color characters other than emoji. We give two examples (sometimes fonts are mentioned on the ConTEXt mailing list).

```
\definefontfeature[whatever]
 [default,color:svg][script=latn,language=dflt]
\definefont[TestF]
 [file:Abelone-FREE.otf*whatever           @13bp]
\definefont[TestG]
 [file:Gilbert-ColorBoldPreview5*whatever @13bp]
\definefont[TestH]
 [file:ColorTube-Regular*whatever          @13bp]
```

COMING BACK TO THE USE OF TYPEFACES IN ELECTRONIC PUBLISHING: MANY OF THE NEW TYPOGRAPHERS RECEIVE THEIR KNOWLEDGE AND INFORMATION ABOUT THE RULES OF TYPOGRAPHY FROM BOOKS, FROM COMPUTER MAGAZINES OR THE INSTRUCTION MANUALS WHICH THEY GET WITH THE PURCHASE OF A PC OR SOFTWARE. THERE IS NOT SO MUCH BASIC INSTRUCTION, AS OF NOW, AS THERE WAS IN THE OLD DAYS, SHOWING THE DIFFERENCES BETWEEN GOOD AND BAD TYPOGRAPHIC DESIGN. MANY PEOPLE ARE JUST FASCINATED BY THEIR PC'S TRICKS, AND THINK THAT A WIDELY—PRAISED PROGRAM, CALLED UP ON THE SCREEN, WILL MAKE EVERYTHING AUTOMATIC FROM NOW ON.

Of course one can wonder about the readability of these fonts and unless one used random colors (which bloats the file immensely) it might become boring, but typically such fonts are only meant for special purposes.

The previous font is the largest and as a consequence also puts some strain on the viewer, especially when zooming in. But, because viewers cache Type 3 shapes it's a one-time penalty.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely—praised program, called up on the screen, will make everything automatic from now on.

This font is rather lightweight. Contrary to what one might expect, there is no transparency used (but of course we do support that when needed).

coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a pc or software. there is not so much basic instruction as of now as there was in the old days showing the differences between good and bad typographic design. many people are just fascinated by their pc's tricks and think that a widelypraised program called up on the screen will make everything automatic from now on.

This third example is again rather lightweight. Such fonts normally have a rather limited repertoire although there are some accented characters included. But they are not really meant for novels anyway. If you look closely you will also notice that some characters are missing and kerning is suboptimal.

I considered testing some more fonts but when trying to download some interesting looking ones I got a popup asking me for my email address in order to subscribe me to something: a definite no-go.

I already mentioned that we have a built-in converter that goes from SVG to MetaPost. Although this article deals with fonts, the following code demonstrates that we can also access such graphics in Metafun itself. The nice thing is that because we get pictures, they can be manipulated.

```
\startMPcode
    picture p ; p :=
      lmt_svg [filename="mozilla-svg-001.svg"] ;
    numeric w ; w := bbwidth(p) ;
    draw p ;
    draw p xscaled -1 shifted (2.5*w,0);
    draw p rotatedaround(center p,45)
           shifted (3.0*w,0) ;
    draw image (
        for i within p : if filled i :
            draw pathpart i withcolor green ;
        fi endfor ;
    ) shifted (4.5*w,0);
    draw image (
        for i within p : if filled i :
            fill pathpart i withcolor red
                 withtransparency (1,.25) ;
        fi endfor ;
    ) shifted (6*w,0);
\stopMPcode
```

This graphic is a copy from a glyph from an emoji font, so we have plenty of such images to play with. The above manipulations result in:

Now that we're working with MetaPost we may as well see if we can also load a specific glyph, and indeed this is possible.

```
\startMPcode
  picture lb, rb ;
  lb := lmt_svg
        [ fontname = "Gilbert-ColorBoldPreview5",
          unicode = 123 ] ;
  rb := lmt_svg
        [ fontname = "Gilbert-ColorBoldPreview5",
          unicode = 125 ] ;
  numeric dx ; dx := 1.25 * bbwidth(lb) ;
  draw lb ;
  draw rb shifted (dx,0) ;
  pickup pencircle scaled 2mm ;
  for i within lb :
      draw lmt_arrow [
          path        = pathpart i,
          pen         = "auto",
          alternative = "curved",
          penscale    = 8
      ]
          shifted (3dx,0)
          withcolor "darkblue"
          withtransparency (1,.5) ;
  endfor ;
  for i within rb :
      draw lmt_arrow [
          path        = pathpart i,
          pen         = "auto",
          alternative = "curved",
          penscale    = 8
      ]
          shifted (4dx,0)
          withcolor "darkred"
          withtransparency (1,.5) ;
  endfor ;
\stopMPcode
```

Here we first load two character shapes from a font. The Unicode slots (which here are the same as the ASCII slots) might look familiar: they indicate the curly brace characters. We get two pictures and use the `within` loop to run over the paths within these shapes. Each shape is made from three curves. As a bonus a few more characters are shown.

It is not hard to imagine that a collection of such graphics could be made into a font (at runtime). One only needs to find the motivation. Of course one can always use a font editor (or Inkscape) and tweak there, which probably makes more sense, but sometimes a bit of MetaPost hackery is a nice distraction. Editing the SVG code directly is not that much fun. The overall structure often doesn't look that bad (apart from often rather redundant grouping):

```
<svg xmlns="http://www.w3.org/2000/svg">
    <path fill="#d87512" d="..."/>
    <g fill="#bc600d">
        <path d="..."/>
    </g>
    <g fill="#d87512">
        <path d="..."/>
        <path d="..."/>
    </g>
    <g fill="#bc600d">
        <path d="..."/>
    </g>
    ...
</svg>
```

In addition to `path`s there can be `line`, `circle` and similar elements but often fonts just use the `path` element. This element has a `d` attribute that holds a sequence of one character commands that each can be followed by numbers. Here are the start characters of four such attributes:

```
M11.585 43.742s.387 1.248.104 3.05c0 0 ...
M53.33 39.37c0-4.484-35.622-4.484-35.622 0 0 ...
M42.645 56.04c1.688 2.02 9.275.043 ...
M54.2 41.496s-.336 4.246-4.657 9.573c0 0 ...
```

Indeed, numbers can be pasted together, also with the operators, which doesn't help with seeing at a glance what happens. Probably the best reference can be found at `https://developer.mozilla.org/en-US/docs/Web/SVG` where it is explained that a path can have move, line, curve, arc and other operators, as well use absolute and relative coordinates. How that works is for another article.

How would the TEX world look like today if Don Knuth had made METAFONT support colors? Of course one can argue that it is a bitmap font generator, but in our case using high resolution bitmaps might even work out better. In the example above the first text fragment uses a font that is very inefficient: it overlays many circles in different colors with slight displacements. Here a bitmap font would not only give similar effects but probably also be more efficient in terms of storage as well as rendering. The MetaPost successor does support color and with MPlib in LuaTEX we can keep up quite well ... as hopefully has been demonstrated here.

◇ Hans Hagen
http://pragma-ade.com

Typesetting the Bangla script in Unicode TEX engines — experiences and insights

Md Qutub Uddin Sajib

Abstract

The typesetting of Bangla (also known as Bengali) script in TEX was first introduced more than 15 years ago through transliteration-based systems. These systems have shortcomings: among others, the source files are harder to read and they require one or two particular Bangla typeface families for typesetting. With the introduction of Unicode-aware TEX engines, such as XETEX, and the emergence of Unicode-compliant free Bangla fonts, new possibilities have evolved. Today both XETEX and LuaTEX, as available in TEX Live 2019, support Bangla typesetting allowing the user to input the text directly with Unicode Bangla fonts in the editor. Although several years have passed since the XETEX system was first seen to work, it is still in a state where the *finest* typographic quality is nearly unachievable for this particular script. Several rendering issues were observed while working with Unicode Bangla fonts in four Unicode-aware TEX engines. Precision typesetting of the Bangla script in Unicode TEX requires attention in terms of fonts, rendering, hyphenation, use of colors, and more.

1 Introduction

The language Bangla, also known as Bengali, is one of the ten most-spoken languages in the world, as reported by *Ethnologue* in its 2019 edition. Native speakers of this language are mainly from Bangladesh, a small but populous country in south Asia. Another good number of native speakers are from the West Bengal state of India. The Bangla script, the written form of the Bangla language, is one of the thirteen major Indic scripts and has made its way into the Unicode Standard. Publishing in this script has a history of many centuries. Like other Indic scripts, typesetting of the Bangla script in TEX has seen several attempts in the last few years but typographic quality has yet to reach a peak.

Apart from beautiful rendering of mathematical contents in TEX, another goal of this typesetting system is the *finest* typographic quality [5]. The same philosophy can be expected in typesetting other scripts, including Bangla. Considering the present-day support of the Bangla script in TEX, this article discusses a few rendering issues, mostly gathered from the author's day-to-day typesetting experiences; it also provides some insights for future development.

2 Scope of this article

Before the Unicode Standard was created to enable the writing of most scripts of the world on computers, the attempts to typeset Bangla script in TEX were confined to ASCII-based transliteration systems. Brief discussions of ASCII- and Unicode-based typesetting of this script are presented in sections 3 and 4. The TEX packages and fonts available today that support Unicode Bangla typesetting are discussed in sections 5 and 6.

It is predictable that most Bangla documents contain at least English, math, and possibly other scripts. In this article, however, we have considered typesetting of the Bangla script only, using the four TEX engines that support the Unicode Standard. This article does not cover the discussion on font selection techniques for different scripts except Bangla. For information on selecting specific fonts for Roman (English) and math along with Bangla, the fontspec package [11] can be consulted.

TEX engines known to support the Unicode Standard are XETEX, LuaTEX, HarfTEX, and LuaHBTEX. The first two are available in TEX Live 2019; the last two via tlcontrib or Akira Kakuto's w32tex and w64tex distributions (`http://w32tex.org/`). We used all four engines to typeset some text of Bangla script to observe the rendering with different fonts (Section 7), hyphenation (Section 9), and use of colors (Section 10). Some development ideas for this particular script are discussed in Section 12.

In this paper, using XETEX means compiling the `.tex` file with xelatex; using LuaTEX, HarfTEX, and LuaHBTEX means compiling the same file with lualatex, harflatex, and luahblatex, respectively. The TEX-specific examples presented here were produced using the TEX Live 2019 distribution on a computer running the GNU/Linux operating system (Slackware 14.2). The HarfTEX and LuaHBTEX engines were installed via tlcontrib, following the instructions at `https://contrib.texlive.info/`.

3 ASCII-based transliteration systems

ASCII-based systems to typeset the Bangla script in TEX were first seen to work more than 15 years ago. The two transliteration schemes known to support the Bangla script are `ITRANS` (**I**ndian languages **TRANS**literation) by Avinash Chopde and the Velthuis system by Frans Velthuis. Both of these schemes were primarily developed for the Devanagari script. Later, the schemes were adapted to typeset the Bangla script in TEX.

The typeface families that work with `ITRANS` include the "SonarGaon" (`sgaon`) fonts by Anisur Rahman [7] and "AroSgaon" fonts by Muhammad

কে লইবে মোর কার্য, কহে সন্ধ্যা রবি
শুনিয়া জগৎ রহে নিরুত্তর ছবি ।
মাটির প্রদীপ ছিল, সে কহিল, স্বামি
আমার যেটুকু সাধ্য করিব তা আমি ।
— রবিন্দ্রনাথ ঠাকুর

```
{\bn ke la{}ibe mor kaarya, kahe sandhyaa rabi
"suni.yaa jagaT rahe niruttar chabi |
maa.tir pradiip chila, se kahila, sbaami
aamaar ye.tuku saadhya kariba taa aami |
                        -- rabindranaath .thaakur}
```

Figure 1: Typesetting of Bangla script in TEX with a transliteration system using the METAFONT-generated "Bengali" fonts (source: [7]).

কে লইবে মোর কার্য, কহে সন্ধ্যারবি।
শুনিয়া জগৎ রহে নিরুত্তর ছবি।
মাটির প্রদীপ ছিল, সে কহিল, স্বামী,
আমার যেটুকু সাধ্য করিব তা আমি।
– রবীন্দ্রনাথ ঠাকুর

```
কে লইবে মোর কার্য, কহে সন্ধ্যারবি।\\
শুনিয়া জগৎ রহে নিরুত্তর ছবি।\\
মাটির প্রদীপ ছিল, সে কহিল, স্বামী,\\
আমার যেটুকু সাধ্য করিব তা আমি।

\vskip6pt\raggedleft
-- রবীন্দ্রনাথ ঠাকুর
```

Figure 2: Typesetting of Bangla script in XETEX with Unicode fonts (spelling of a few words, as they appeared in Figure 1, were corrected following [14]).

Masroor Ali [1]. The latter was available with its METAFONT sources and was replaced by the Type 1 "ITXBengali" fonts of Shrikrishna Patil in ITRANS. The bwti (Bengali Writer TEX Interface) package by Abhijit Das included METAFONT-generated "Bengali" fonts and worked in TEX through a special interface.

The bengali package [8] by Anshuman Pandey uses the Velthuis transliteration scheme instead of ITRANS. It uses the latest version of Das's "Bengali" fonts for typesetting. The bangtex package [6] by Palash Baran Pal includes class files and METAFONT sources for its "Bangla" fonts. The Type 1 fonts for this package were created by Ananda Kumar Samaddar [6] and are included in the bengali-omega package [10] of Lakshmi K. Raut. The latter uses the Velthuis transliteration scheme. It also supports Unicode-based input but would convert the Unicode text into the transliteration scheme for typesetting.

The transliteration-based systems require the user to input Bangla text in a specific scheme with fonts from the Roman script. Then the text would be processed with preprocessors for typesetting in TEX. Although these systems work, the source file is harder to read (Figure 1). They seem to use the "Bengali" fonts (from bwti) or "Bangla" fonts (from bangtex) to typeset the document. The typographic quality of these fonts may not be comparable with fonts we see in modern Bangla publications.

4 Unicode-aware TEX engines

With the introduction of XETEX and LuaTEX around 2007, and the fontspec package for selecting TrueType and OpenType fonts, typesetting of Bangla in TEX using Unicode fonts became a reality. Today, a good number of Unicode-compliant Bangla fonts are freely available that work with these engines.

To start, one needs a keyboard layout that supports the input of Unicode Bangla characters in an editor. In most GNU/Linux systems, a keyboard layout called Probhat is available for this purpose. A popular alternative is the *Avro Keyboard*, available for free (https://www.omicronlab.com/index.html), which can be installed in GNU/Linux, Mac OS X, and Windows systems. In the emacs editor, as of version 26.2, three layouts are available, namely bengali-inscript, bengali-itrans, and bengali-probhat.

None of the Unicode Bangla keyboard layouts available today were designed with TEX users in mind; hence one may need to switch the layout frequently in order to type special TEX characters (\, %, &, etc.). An appropriate font containing the Bangla script has to be set via the fontspec package (details in Section 6). Then, upon processing the .tex file with xelatex, lualatex, harflatex, or luahblatex one gets the typeset document.

The Unicode-based systems in TEX for this script have many advantages over the older systems. For example, the source file is now easy to read (Figure 2, right versus Figure 1, right). In addition, any font that contains the glyphs for this script can be used for typesetting. However, the current situation is not free from shortcomings. The verbatim text in Figure 2 (right), which should read "\vskip6pt\raggedleft", is unreadable because the font used there contains glyphs only from the Bangla script. Other shortcomings that we have observed are discussed in the following sections.

5 LATEX packages for Unicode Bangla

The polyglossia package by François Charette [2] is designed to provide support for typesetting Bangla script, along with other scripts, using suitable Unicode fonts and TEX engines. It provides a style file (begalidigits.sty) for this script that translates the Arabic numerals into Bangla numerals. The language definition file (gloss-bengali.ldf) implements the Bangla numerals in LATEX counters. It also provides Bangla translation for the names of LATEX sections and counters, and for the Gregorian calendar months.

The latexbangla package by Adib Hasan [3] introduces some control sequences to select Bangla fonts. To our knowledge, there are no Unicode Bangla fonts designed to be used with TEX. As such, the package sticks with the limited fonts available today and makes bold, slanted, and monospaced text using fonts from different designers. It uses the AutoFakeBold and AutoFakeSlant features to produce *fake* styles. As a result, the typeset document looks something passable but not of great aesthetic taste. The fonts used

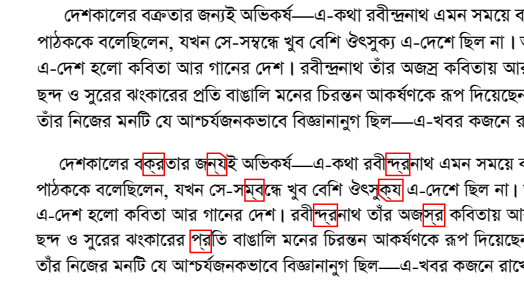


Figure 3: Rendering of Bangla script in XeTeX using Free Serif fonts: top: using MiKTeX 2.8; bottom: using TeX Live 2019 (red boxes indicate wrong rendering).

in this package are not available in TeX Live 2019. Besides, its dependence on the ucharclasses package [4] makes it unusable with other engines than XeTeX.

6 Unicode-compliant Bangla fonts and TeX

TeX Live 2019 comes with the gnu-freefont package which contains Unicode fonts in both TTF and OTF formats and covers a wide range of the Unicode character set. Fonts for the Bangla script are available in serif and sans-serif versions, in regular and slanted styles. Unfortunately, no bold or bold italic fonts are available for this particular script. Figure 3 shows the rendering of a few lines of Bangla script using Free Serif fonts with xelatex. In this figure, the typeset text on top is from the author's own typesetting for a book [9] that was compiled in 2012 with xelatex using the MiKTeX 2.8 distribution. The same piece of text was found producing a bit different output when compiled with xelatex using TeX Live 2019; to be specific, a few conjunct characters and ligatures are incorrectly rendered. This rendering problem, whether it concerns the Free Serif fonts or the xelatex program, needs to be fixed in the future.

Besides the Free Serif fonts in TeX Live 2019, the noto font family from Google (`https://www.google.com/get/noto/`) includes Noto Serif Bengali and Noto Sans Bengali fonts in TTF format. The serif version includes the regular and bold styles while the sans-serif version contains *seven* other styles. All these fonts can be used to typeset Unicode Bangla in TeX but they must be downloaded and set up correctly so that TeX finds them. The OTF version of this font family *is* available in TeX Live 2019 but it does not include the fonts for Bangla script.

A good number of Unicode-compliant Bangla fonts are available today and can be downloaded for free. The *Avro Keyboard* website has a dedicated page for such fonts (`https://www.omicronlab.com/bangla-fonts.html`); the *Ekushey* project also has a page (`http://ekushey.org/index.php/page/33`) for this purpose. Most Unicode Bangla fonts available today, except the two mentioned above, are not available in slanted, italic, bold, etc., styles. This is probably due to the fact that those fonts were not designed with professional publication in mind; also, not with TeX users in mind. When using XeTeX, the AutoFakeBold and AutoFakeSlant features can be used and the result is somewhat acceptable. In LuaTeX, HarfTeX, and LuaHBTeX, however, these features are not supported.

Besides the lack of publication-quality Unicode Bangla fonts, the rendering of Bangla script in Unicode TeX engines needs deeper attention. To experiment with the four engines available today, we selected three fonts to typeset the same piece of texts. The first one is the Free Serif font available in TeX Live 2019, second one is the Noto Serif Bengali, and third one is the Lohit Bengali font. This last is available in most GNU/Linux distributions; otherwise, it can be downloaded from the *Ekushey* font page mentioned above. The font setup used for all examples in this article is given below. The Noto Sans Bengali font was used in Figure 2 to typeset the verbatim text. Considering the x-height of Free Serif fonts as "normal", other fonts were scaled accordingly to get the identical typeset output. This font setup works with xelatex, lualatex, harflatex, and luahblatex:

```
\usepackage{fontspec}
\usepackage{ifxetex}
%
\newfontfamily{\freeserifbn}{FreeSerif.ttf}
  [Script=Bengali, Ligatures=TeX]
\newfontfamily{\notoserifbn}
  {NotoSerifBengali-Regular.ttf}
  [Script=Bengali, Scale=0.85, Ligatures=TeX]
\newfontfamily{\notosansbn}
  {NotoSansBengali-Light.ttf}
  [Script=Bengali, Scale=1, Ligatures=TeX]
\ifxetex
\newfontfamily{\lohitbn}{lohit_bn.ttf}
  [Path=/usr/share/fonts/TTF/,
  Script=Bengali, Scale=0.82, Ligatures=TeX]
\else
\newfontfamily{\lohitbn}{lohit_bn.ttf}
  [Script=Bengali, Scale=0.82, Ligatures=TeX]
\fi
```

7 The DOTTED CIRCLE in rendering the Bangla script

The glyph named DOTTED CIRCLE is in the Geometric-Shapes Unicode block, assigned the character code U+25CC. Thus it can be typeset with the TeX command `\char"25CC`, using the font setup above

and compiling the input file with xelatex, lualatex, harflatex or luahblatex to get: ◌ (here with the Free Serif font). In non-Roman scripts, the DOTTED CIRCLE can be used as the base character to typeset a combining mark [15] which would otherwise be combined with a real base character of a script.

In the Bangla script, a *vowel sign* or the short form of a vowel (known as *kaar*) replaces the glyph of that vowel when the vowel is followed by (or modifies) a consonant (base character). For example, the vowel "আ" (BENGALI LETTER AA) has the short form "া" (BENGALI VOWEL SIGN AA). When the consonant "ক" (BENGALI LETTER KA) is modified by the vowel আ, the vowel sign (া) replaces the actual vowel and the consonant-vowel conjunct is typeset as "কা". Similarly, several consonants have short forms (known as *phalas*); they replace their actual glyphs (bases) when two or more consonants are combined to produce a conjunct character.

The *kaars* and *phalas* in Bangla script can collectively be called combining marks. Since different *kaars* and *phalas* have different positions to stand with a base character, the DOTTED CIRCLE can be helpful to visualize the actual position of a combining mark when a *kaar* or *phala* is typeset independently.

In Figure 4, the most common vowel signs are shown as independently typeset combining marks (first row of a pair) and combined with the consonant ক (second row of a pair). The example is shown using three fonts, Free Serif, Noto Serif Bengali, and Lohit Bengali, compiling each with our four Unicode TEX engines. As seen in this figure, a DOTTED CIRCLE unexpectedly appears when a *vowel sign* is typeset with xelatex; this is not the case with lualatex, harflatex or luahblatex. On the other hand, when vowels are combined with a consonant, xelatex seems to render the script correctly; the other three engines fail except for few consonant-vowel combinations with the Lohit Bengali font. In the case of consonant-consonant combinations, xelatex failed for specific combinations with the Free Serif font (Figure 4, middle column, top), although it worked for other combinations (not shown in figure).

Problem arises when one intends to typeset a vowel sign or other combining marks *without* the DOTTED CIRCLE. It is particularly necessary when these signs are taught to children. Ideally, in TEX, one should be able to typeset the combining marks (*kaars* or *phalas*) independent of the DOTTED CIRCLE. We say *ideally* because: (i) when a consonant combines with a vowel sign, the DOTTED CIRCLE disappears implying the presence of these signs as independent glyphs in the current font; (ii) as described in the Unicode Standard and mentioned previously, the DOTTED CIRCLE *can be* used as a base character to a combining mark; and (iii) in a font viewer like FontForge, the vowel signs are indeed found as independent glyphs. Therefore, rendering of the combining marks, especially the vowel signs, in xelatex with a DOTTED CIRCLE even when it is undesirable can be considered as a bug (more in Section 11).

Figure 4: Typesetting of the vowel signs, independently and as a conjunct with a consonant (left column), and a consonant-consonant conjunct character (middle column) in four engines using the Free Serif (top), Noto Serif Bengali (middle), and Lohit Bengali (bottom) fonts.

8 Rendering of Bangla script in HarfBuzz, word processors, and elsewhere

In order to understand the appearance of DOTTED CIRCLE in typesetting the combining marks, it is logical to look into the output produced by the text rendering stack HarfBuzz (`https://www.freedesktop.org/wiki/Software/HarfBuzz/`). This software is known to work behind the XETEX engine as well as in many word processors, text editors, web browsers, and probably elsewhere. Two modules hb-view and hb-shape are available in the HarfBuzz Indic Shaper and can be used on Unix systems to get the rendered output of a Unicode script. As shown in Figure 5, HarfBuzz produces the same result for different fonts as we have seen with xelatex in Figure 4.

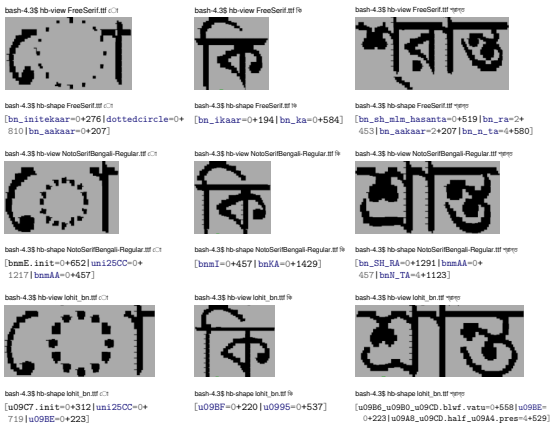


Figure 5: Rendering of Bangla script in HarfBuzz using different fonts in a GNU/Linux shell.

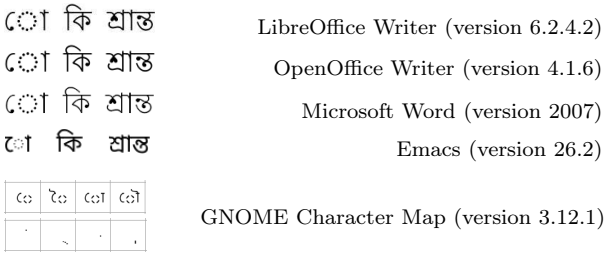


Figure 6: Rendering of the Unicode Bangla in word processors and emacs (top); Bengali (Bangla) and Hebrew scripts in GNOME Character Map (bottom).

When Unicode Bangla characters are input in word processors or the emacs editor using a compatible keyboard layout, the rendering (Figure 6) can be seen to be the same as the HarfBuzz-generated output (Figure 5). In the address bars of popular web browsers, as tested in Firefox and Chromium, the same kind of rendering was also observed (not shown in figure). The GNOME Character Map, however, displays the combining marks of Bangla and Hebrew scripts differently in terms of the DOTTED CIRCLE (Figure 6).

The examples in figures 5 and 6 imply that HarfBuzz is responsible for the unexpected appearance of DOTTED CIRCLE in XeTeX when the vowel signs are typeset independently. This assumption is also supported by the fact that the LuaTeX engine produces expected results in terms of DOTTED CIRCLE (first row of second pairs in Figure 4), as it does not depend on HarfBuzz for rendering. The HarfTeX and LuaHBTeX rendering (third and fourth pairs in Figure 4), however, seems puzzling as they are known to use HarfBuzz but output LuaTeX-like rendering, although XeTeX-like rendering could be expected.

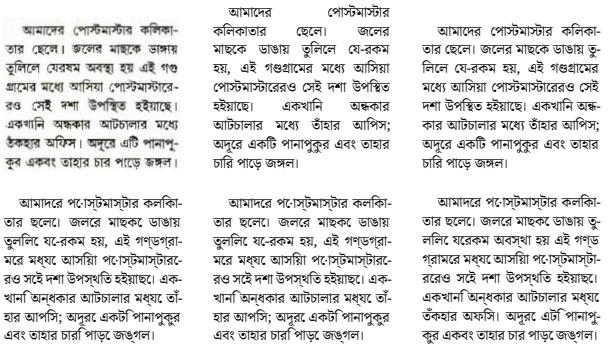


Figure 7: Hyphenated text from [12, p. 391] (top left), xelatex with no hyphenation (top middle). polyglossia-generated hyphenation in: xelatex (top right), lualatex (bottom left), harflatex (bottom middle), and luahblatex (bottom right). (The spelling of a few words was corrected following [13]).

9 Dealing with hyphenation

In modern-day Bangla publications, hyphenation is hardly seen, either because of technical limitations or lack of interest. Fortunately, the polyglossia package supports hyphenation for Bangla script which seems to work well (Figure 7). All four Unicode TeX engines were found to be working with hyphenation, although the lualatex, harflatex, and luahblatex have rendering issues as discussed previously. However, the hyphenation rule `hyphenmins={2,2}` as found in the `gloss-bengali.ldf` file is probably too low for this script. It was also found that any changes made in this file, e.g., `hyphenmins={3,3}` has the desired effect but using the same in a `.tex` file has no effect in the hyphenation pattern.

10 Typesetting with color

Use of colors in text can significantly improve the visual as well as readability for particular types of contents. Colorful text can be essential in books written for children. For Bangla, sometimes it is desirable to typeset different parts of a conjunct character in different colors to help children learn and recognize them with ease. A good example is to flag a *kaar* in a different color than its consonant base, as can be seen in textbooks for children (Figure 8, first column, first and second row). Such use of color was not found to be working with xelatex; the lualatex, harflatex and luahblatex were found to be working in a few combinations (Figure 8, second column, first and second row) but failing in other cases. Similarly, typesetting a ligature with its different parts flagged in different colors was found to be not possible (Figure 8, third row). The code to typeset the colorful

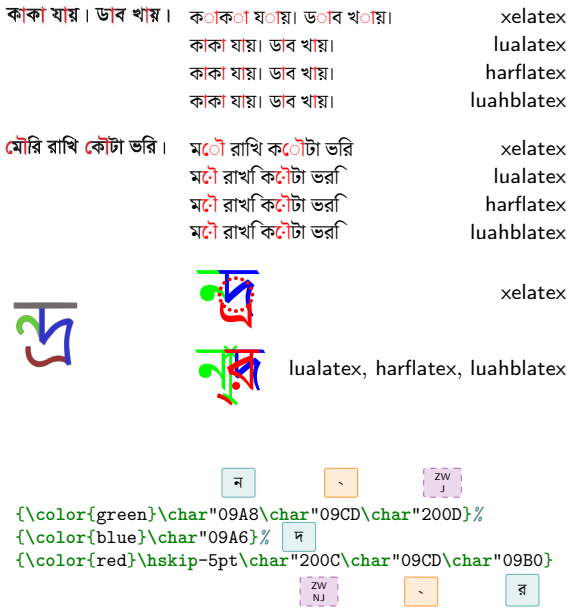


Figure 8: Use of different colors for different parts of a conjunct or ligature is a challenge; left: example from `http://tiny.cc/bdnctb-classone` (top and middle rows) and `http://tiny.cc/4xly9y` (bottom row); middle column: TEX output.

ligature (Figure 8, fourth row) is somewhat complex and may not be very useful in real life typesetting.

11 The DOTTED CIRCLE mystery, revisited

To put it simply, one should be able to typeset any *glyph* of a font independently, that is, without the DOTTED CIRCLE as a base when expected. As seen in previous examples, a few glyphs cannot be typeset independently. To understand this behavior in TEX, we try to deconstruct the rendering of DOTTED CIRCLE using the TEX primitive `\char"` and typesetting some Unicode character codes from Bangla script (Figure 9). In this example, the appearance of DOTTED CIRCLE (`\char"25CC`) can be considered as *unusual* because a single call of `\char"09BE` is seen to typeset both "◌া" (`\char"25CC`) and "া" (`\char"09BE`). Also, both `\char"09BE` and `\char"25CC\char"09BE` commands are seen to produce the same typeset output. Other examples in this figure further suggest that the rendering may not be called satisfactory.

The apparent problem that DOTTED CIRCLE *cannot* be separated even when it is undesired in typesetting Indic scripts was previously reported as a bug in LibreOffice (`http://tiny.cc/bsebez`), and also mentioned on the xetex mailing list (`http://tiny.cc/atgbez`). The LibreOffice page has declared this issue as not a bug while no conclusion was found on the list. However, Khaled Hosny gave

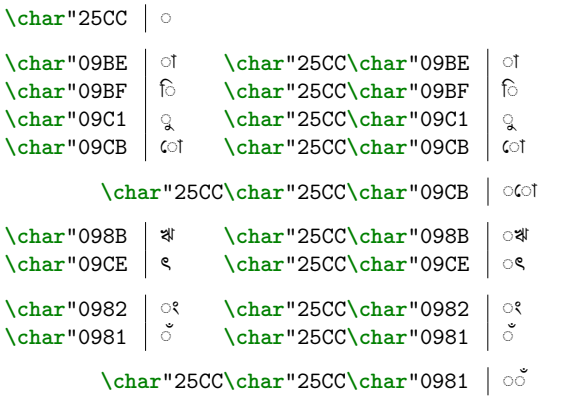


Figure 9: Deconstructing the DOTTED CIRCLE in XETEX with base characters from Bangla script.

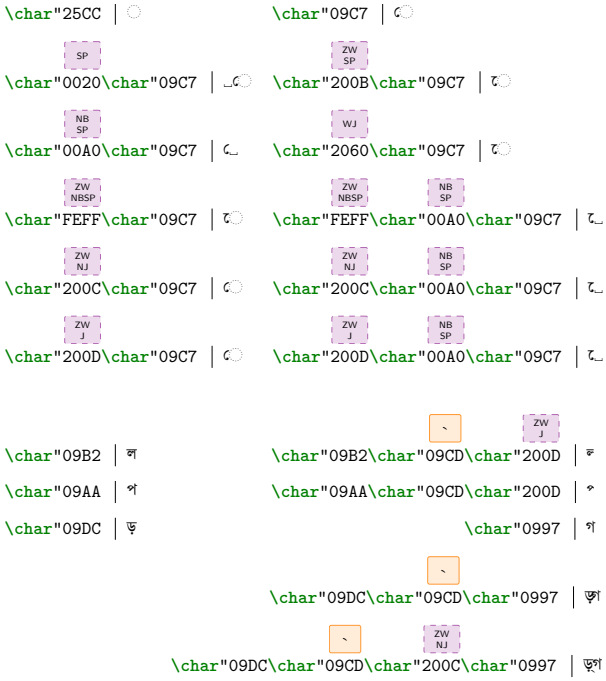


Figure 10: Deconstructing the DOTTED CIRCLE to typeset combining marks and special conjuncts in XETEX with Bangla script.

advice on the LibreOffice page to use a SPACE or NO-BREAK SPACE before the given glyph when DOTTED CIRCLE is undesired.

The NO-BREAK SPACE (NBSP) was found to work to "remove" the DOTTED CIRCLE when used before a combining mark while use of a SPACE (SP), as predicted, did not work (Figure 10, top). However, this trick for removing DOTTED CIRCLE may not be acceptable because it actually *replaces* the DOTTED CIRCLE with a space (shown with a "␣" in figure).

Other combinations were tested, using ZERO WIDTH SPACE (ZWSP), WORD JOINER (WJ), ZERO

WIDTH NO-BREAK SPACE (ZWNBSP), ZERO WIDTH NON-JOINER (ZWNJ), and ZERO WIDTH JOINER (ZWJ). The result is interesting as we get ৎ্ and ৎ্ (notice the horizontal stroke on top, known as *maatra* in Bangla script) using different combinations but the same character code (U+09C7). The ZWNJ and ZWJ characters were found useful in typesetting short forms of consonants and special conjunct characters (Figure 10, bottom).

12 What next?

In order to achieve the *finest* typographic quality in Bangla script, several things can be taken into consideration. The rendering issues especially with the DOTTED CIRCLE in XeTeX, and conjunct characters and ligatures in other engines may take priority. Contact can be made with the HarfBuzz developers for this purpose. For now, because rendering issues are observed in XeTeX, LuaTeX, HarfTeX, and LuaHBTeX, it is probably necessary to experiment with all these engines. Eventually we may want to settle on one particular engine.

The Noto Bengali fonts, both serif and sans-serif, can be included in the future versions of TeX Live. This would allow the users to try Unicode-aware TeX engines with at least two font families including the already existing Free Serif and Free Sans fonts. Eventually, a dedicated font family for the Bangla script should be designed especially with TeX users in mind and a supporting macro package developed. Supporting only the fontspec package at the primary stage would be fine; integration with the polyglossia package may come next.

A keyboard layout can be designed for the purpose of Unicode Bangla character input making it emacs- and TeX-friendly. In this design, keys for the special TeX characters (\, %, &, etc.) can be retained, so that these keys can be used to format Bangla text without having to switch keyboard layouts. Keys for the Unicode characters NO-BREAK SPACE, ZERO WIDTH NON-JOINER, ZERO WIDTH JOINER, ZERO WIDTH NO-BREAK SPACE, etc., should also be in the layout design since these characters can be helpful in typesetting combining marks and special conjuncts. As these characters are not visible in the editor when input directly, they can be even better input with newly-defined TeX macros.

13 Conclusion

The present-day Bangla publishing industry is mostly not using the fine typographic power of TeX. The reasons behind this are many, of which a few are discussed here. Interest in solving those issues has been seen in recent years. Although use of TeX in typesetting Bangla fiction books might be a bigger challenge, mostly due to non-TeXnical reasons, a good number of science books can be expected in the future. For this to happen, the current limitations of Unicode TeX engines and fonts need to be addressed. The few insights we were able to bring into light in this article may lead us to the beginning of the *finest* typographic quality in Bangla publishing.

References

[1] M. M. Ali. *AroSgaon (More SGAON) 2.1*, 1996.

[2] F. Charette. *Polyglossia: An Alternative to Babel for XeLaTeX and LuaLaTeX*, 1.44 edition, 2019.

[3] A. Hasan. *The LaTeXbangla Package: Enhanced LaTeX integration for Bangla*, 0.2 edition, 2016.

[4] M. P. Kamermans. *ucharclasses*, 2017.

[5] D. E. Knuth. *The TeXbook*, vol. A of *Computers & Typesetting*. Addison–Wesley Publishing Company, Massachusetts, 2012.

[6] P. B. Pal. Bangtex: A package for typesetting documents in Bangla using the TeX/LaTeX systems, 2002. `http://www.saha.ac.in/theory/palashbaran.pal/bangtex/bangtex.html`

[7] A. Pandey. Typesetting Bengali in TeX. *TUGboat* 20(2):119–126, 1999. `https://tug.org/TUGboat/tb20-2/tb63pand.pdf`

[8] A. Pandey. *Bengali for TeX*, 2.0 edition, 2002.

[9] A. M. H. Rashid. *Baanglaay Renesnaar Pothikrit: Rabindranath O Chaar Bangali Bigynani (Trailblazer of Renaissance in Bengal: Rabindranath and Four Bengali Scientists)*. Nabajuga Prokashani, Dhaka, 2015. in Bangla.

[10] L. K. Raut. *Typesetting Bengali in Ω using Velthuis Transliteration or Unicode Text*, 2006.

[11] W. Robertson. *The fontspec package: Font selection for XeLaTeX and LuaLaTeX*, 2.7c edition, 2019.

[12] F. G. E. Ross. *The Evolution of the Printed Bengali Character from 1778 to 1978*. PhD thesis, School of Oriental and African Studies, University of London, 1988. `https://eprints.soas.ac.uk/29311/1/10731406.pdf`

[13] R. Tagore. The complete works of Rabindranath Tagore: Stories. Retrieved: 2 August 2019. `http://tiny.cc/tagore-postmaster`

[14] R. Tagore. The complete works of Rabindranath Tagore: Verses. Retrieved: 15 July 2019. `http://tiny.cc/tagore`

[15] The Unicode Consortium. *The Unicode Standard, Version 12.1.0*. The Unicode Consortium, Mountain View, CA, 2019. `https://unicode.org/versions/Unicode12.1.0/`

◇ Md Qutub Uddin Sajib
China University of Geosciences, Wuhan
388 Lumo Road, Wuhan 430074, China
`qsajib71 (at) gmail dot com`
ORCID 0000-0002-7090-7981

Typographers' Inn

Peter Flynn

CV/Résumé layouts

If you use LaTeX for your writing, it's pretty much a no-brainer to use it for other classes of document: presentations (slides), business cards, leaflets... and your CV or résumé.

The CV topic on CTAN has over 30 related packages, including several from the Koma bundle, and the europasscv package which implements a format which is becoming increasingly important in the EU[1] (see Figure 1).

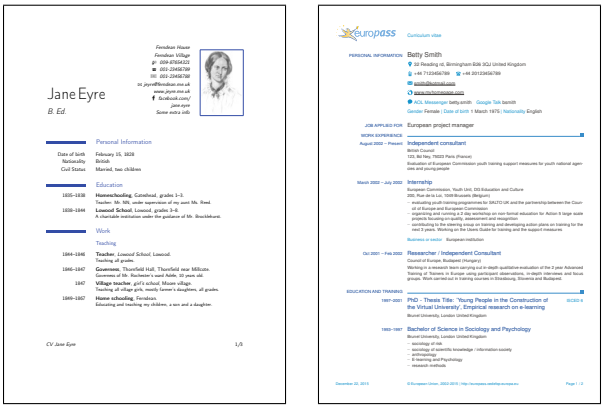


Figure 1: One of the Koma CVs (L) and the EuroPass CV (R) layouts

Sometimes you might need the classical format with label/value pairs, as in the moderncv package; but you might prefer the limecv package, for example, which uses icons, buttons, and bar-graph indicators to brighten up your image (see Figure 2).

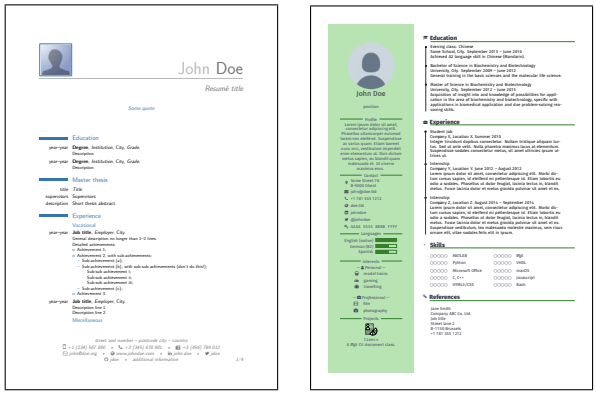


Figure 2: The Modern CV (L) and the Lime CV (R) layouts

[1] europass.cedefop.europa.eu/documents/curriculum-vitae

The two main choices the authors provide are the choice of layout and the choice of typeface. It's largely an aesthetic decision — assuming the actual *content* of your résumé is attractive — but you do need to consider your target audience: what may be required to encourage a City bank to employ you may be very different from what would help a hot design agency to take you on.

Off CTAN, there's also Jan Küster's jankapunkt package[2] which has several variant layouts, including a full-on 'infographics' format, and the wonderful Hipster CV[3] from the LaTeX Ninja[4] (see Figure 3).

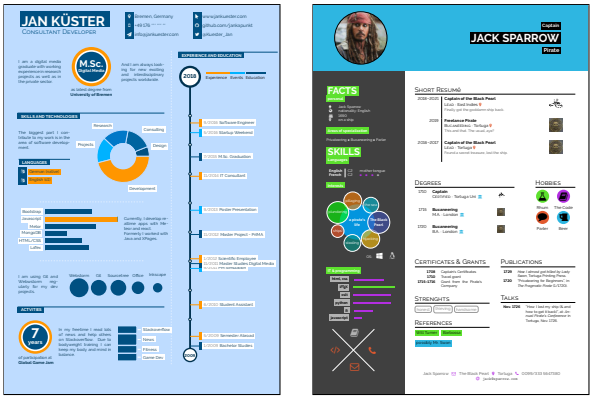


Figure 3: The 'infographics' layout from the jankapunkt package and the Hipster CV from the LaTeX Ninja

The typographic requirements of a résumé entirely depend on who is going to see it and how they will read it. In a traditional environment which follows common HR procedures there may be a triage panel who will merely check that you have fulfilled the basic requirements, and throw out any which clearly fail to meet them. Human Resources may then comb through them and weed out borderline cases. Someone then creates a shortlist of the top few, and if you're in them, you get an interview.

But increasingly, employers faced with hundreds or thousands of applications for a single job will fail to do even this, and they certainly won't write to acknowledge your CV or say how sorry they are that you didn't make the cut. Online submission and automated analysis simply makes it easier to reject the 99.99% *en masse*, reading the text from a PDF file using an AI system, matching what is there against the posted requirements.[5]

[2] github.com/jankapunkt/latexcv
[3] github.com/latex-ninja/hipster-cv
[4] latex-ninja.com/
[5] See examples at http://www.resumehacking.com/ready-for-automated-resume-screening and https://www.vettd.ai/landing/resume-processing.

Before this, large employers with a bulk applicant problem would have to type in the baseline data from each CV, and then use a specially-written HR filter in a statistics or database system to report on the likely candidates.

All this means you really have to stand out from the crowd and make it easier for the employer to see you and then pick *you*. Like so much else in typesetting, there are some simple rules for the continuous text parts:

- **readability**: no long lines, keep the type size above 11pt, and don't use fancy fonts;
- **consistency**: if you do something one way once, do all occurrences of the same thing the same way every time;
- **logic**: follow a pattern or sequence so that one thing leads naturally to the next.

You are trying to construct a train of thought in the reviewers' heads which will lead them to pick you. For a position in a design field, it's likely that the appearance of a résumé will be taken into account, but if the job is computer or office related, it's probably much less likely. At this stage, making it hard to read for the sake of looking cute is probably best avoided.

Good luck with the job![6]

Reversed apostrophes (III)

Way back in 1999, I wrote a short note about responses to a query I had raised on TYPO-L about the 'reversed quotation mark' [1, p. 344]. This is an open-quote character that has been reflected horizontally so that it is a mirror-image of the regular close-quote character, like ‘ and ’.

It wasn't clear then, and still isn't now, what this is done for, except perhaps as a misguided attempt at symmetry, and I have mentioned it again on and off, but had to eat my own dogfood when it turned out to be *much* older than I thought [2, p. 136].

Figure 4: Reversed apostrophe in neon sign (red intensified for clarity)

The disease is still there, though, and it has spread like a virus, and consequently mutated. It's now infecting the apostrophe, and inverting it instead of reflecting it (actually, just using an open-quote where a close-quote should be): I spotted the example in Figure 4 in Budapest this summer.

Afterthought

On Twitter, Julie O'Leary (`@julieoleary90`) posted an interesting photograph on 15th June with the comment:

> More Very interesting mugs at the Ballyanly community centre tonight post 4 mile run. Anyone ever heard of the TeX Users Group meeting in Europe held in @UCC in Sept 1990 where they gave out custom mugs with wolves dressed as leprechauns?

Figure 5: A stray TEX90 mug sighted in the wild

Journalist and historian of the Irish Revolution Niall Murray (`@niallmurray1`) was quick off the mark:

> Read all about it: here's a @tonyleen report in the @irishexaminer [September 11, 1990].[7] If ya wanna join @TeXUsersGroup, their 2019 conference is in Palo Alto, California in August.

Thank you both for the eagle eyes and the comments.

References

[1] P. Flynn. Typographers' Inn — Reversed quotes. *TUGboat* 20(4), Dec 1999. `tug.org/TUGboat/tb20-4/tb65flyn.pdf`

[2] P. Flynn. Typographers' Inn — Comeuppance. *TUGboat* 25(2), Dec 2004. `tug.org/TUGboat/tb25-2/tb81inn.pdf`

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

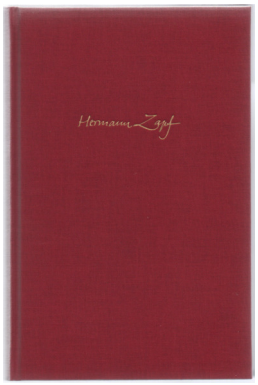
[6] My thanks to Barbara Beeton for her excellent suggestions in review.

[7] `https://pbs.twimg.com/media/D9D1IWJWkAAMPmI.jpg`

Book review: *Hermann Zapf and the World He Designed: A Biography*, by Jerry Kelly

Barbara Beeton

Jerry Kelly, *Hermann Zapf and the World He Designed: A Biography*. The Grolier Club, New York, 2019, hardcover, 364 pp., US$48.00, ISBN 978-1605830827.



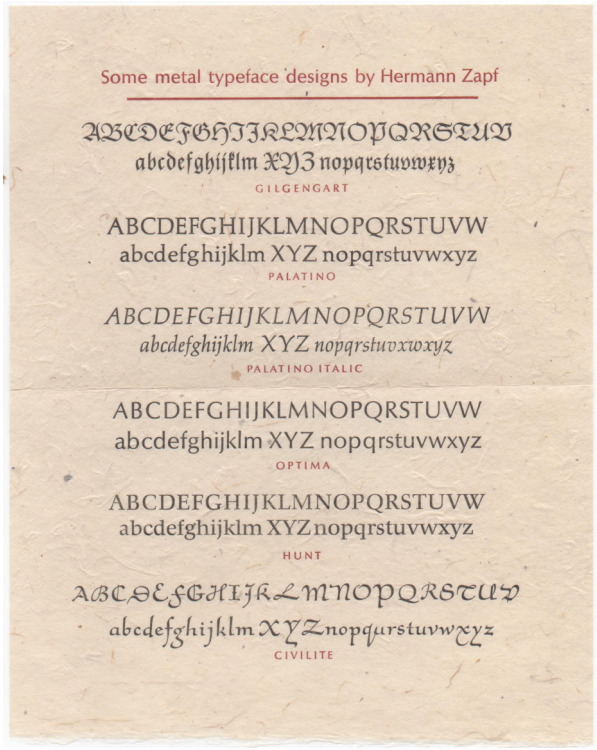
This book, the first comprehensive biography of Hermann Zapf (1918–2015), was published to accompany the Grolier Club exhibition "Alphabet Magic: A Centennial Exhibition of the Work of Hermann & Gudrun Zapf", curated by Jerry Kelly and Steven Galbraith. The exhibition (February 20 – April 27, 2019 [1]) celebrated the centenary of Zapf's birth.

The volume is handsomely bound in blue cloth stamped in gold. (A deluxe edition, obtained through a Kickstarter campaign,[1] is instead bound in red.) Zapf's signature appears on the front cover, with the elegant swash Z that instantly identifies its creator. The colophon identifies the typefaces used for the text as Palatino nova and Optima nova, the versions of those faces re-implemented by Zapf for electronic technology.

Copious illustrations show not only the many typefaces created by Zapf, but also preliminary designs for some of them, calligraphy specimens (both standalone pieces and book jackets), and some delightful drawings, mostly of flowers and other natural subjects. This is a veritable feast! The current location of these exhibits is often identified in footnotes, providing an itinerary if the reader wishes to inspect the originals. (The deluxe edition includes letterpress samples of some of the metal types, as reproduced here.)

The main narrative is divided into three chronological sections. The first covers the period from

[1] A web page describing a visit to Kelly's house and vast library [2] mentions the campaign. Both this and the Kickstarter page [5] show many illustrations of Zapf's work.



Zapf's birth in 1918 through 1954, the years in which his typefaces were rendered in metal. The period 1952–1975 was the era of phototype. Digital methods became available after that. These are not hard boundaries, and Zapf's other activities aren't so easily subdivided, but the division does permit the table of contents to fit on one page. More easily subdivided are the many appendices, which provide important details in a way that makes it easier to look them up. The index is populated mainly by the names of persons, places, typefaces, ..., namely words that inherently begin with an uppercase letter. It's not easy to look up a concept, unless one happens to remember the name of a person associated with it. For example, there is no entry for "calligraphy".

Jerry Kelly first met Zapf as a student, enrolling in a summer calligraphy class at the Rochester Institute of Technology (RIT) in 1975, and repeating this for seven more years. Kelly was already an accomplished letterpress printer, and in cooperation with the RIT Cary Librarian, undertook to print several tribute works. His friendship with Zapf is not explicated, but is evident throughout the text.

Much of the story of Zapf's artistic development has been told before by Zapf, in his books including *Alphabet Stories* [3], but this biography enlarges the scope and includes some information that Zapf was apparently unwilling to divulge in his own writing.

During the second period, 1952–1975, a great deal of Zapf's design work was commissioned by the Stempel foundry. This period saw the creation of many of his best known typefaces: extensions to the Palatino family, Janson (a revival of a type originally designed by Nicolas Kis, in titling sizes to complement the original text types), Melior (which introduced the shape of Piet Hein's superellipse), and Optima (a revolutionary sans serif with tapered strokes more characteristic of serif types), among others. The initial platform for which these types were designed was the Linotype, which imposed very stringent physical limitations, such as that the same letter in parallel upright and italic fonts must have the same metrics. The later freedom of phototype would permit a rethinking of the consequences of these limitations, and subsequent redesign.

During this period, Zapf also designed numerous books. Although most were published by major German trade publishers, some, in particular his works on typographic subjects, were produced at Stempel's in-house printing office. Zapf developed many contacts among the skilled craftsmen at Stempel, and when several of these craftsmen went out on their own, Zapf turned to his former colleagues to produce work at the highest level of quality. Although the production of the second volume of his *Manuale Typographicum* was an artistic masterpiece, it was not a commercial success, and Zapf would come to regret taking on the role of publisher. Later works on subjects related to type and calligraphy were managed by other publishers.

The third period, 1975–2015, deals with the post-metal era, during which Zapf continued to design books, revised his existing typefaces and developed new ones for new technology, and got more involved with teaching, for a time holding a professorship at the Rochester Institute of Technology. During this period he also, with Peter Karow, developed the *hz*-program, which uses a per-paragraph justification system and carefully modifies the shapes of letters to achieve uniform interword spacing and optimize the consistent appearance of text.

Of particular interest to the present audience is the information regarding Zapf's relation to the TEX world. The principal focus here is on the Euler font (pp. 236–239), illustrated by an early drawing of the Latin and Greek alphabets. Several of the letters are more angular in this drawing than in the alphabets we see today, and the "u"-like shape of the "υ" is said to be a particular request of Knuth. Kelly remarks,

> Personally, I feel Euler would have looked better if they would have stuck with Zapf's original sharper entry and exit strokes, but Zapf acquiesced to Knuth's wishes in this matter. (p. 238)

This overlooks the fact that, in a mathematical context, it is essential that each letter be unambiguously recognizable. In the early showing, it is difficult to distinguish the Latin "v" from the "ν" (nu); these and other similar shapes have been adjusted in the final version.

Also noted is the recognition by DANTE, at their 1999 annual meeting in Heidelberg, when Zapf was named an honorary member. At the 2000 annual meeting Zapf responded with remarks published in *Die TEXnische Komödie* [4]; these remarks were translated for *TUGboat* with an addendum illustrating the proper traditional page layout for a book [6,7]. (The biography inexplicably lists this a second time (p. 323) for 2010: "Honorary Member of the German TeX Users Group (Heidelberg)".)

References

[1] Zapf Centennial Symposium at The Grolier Club, March 20, 2019, accessed 1 Sept. 2019, www.tdc.org/event/zapf-centennial-symposium-at-the-grolier-club/

[2] Pradeep Sebastian, "The master of beautiful letters", *The Hindu*, October 27, 2018, accessed 1 Sept. 2019, thehindu.com/books/the-master-of-beautiful-letters/article25357866.ece

[3] Hermann Zapf, *Alphabet Stories*, Mergenthaler Edition, Linotype GmbH, Bad Homburg Germany / Cary Graphic Arts Press, RIT, Rochester, NY, 2007.
Review by Hans Hagen and Taco Hoekwater, *TUGboat* **28**:2 (2007), 174–176, tug.org/TUGboat/tb28-2/tb89hagen.pdf

[4] Frank Mittelbach, "Laudatio auf Professor Hermann Zapf"; Hermann Zapf, "Meine Zusammenarbeit mit Don Knuth und meine Schriftentwürfe", *Die TEXnische Komödie* **2000**:1, 31–44, archiv.dante.de/DTK/PDF/komoedie_2000_1.pdf

[5] "Hermann Zapf & the World He Designed", Kickstarter, accessed 1 Sept. 2019, www.kickstarter.com/projects/1307403978/hermann-zapf-and-the-world-he-designed

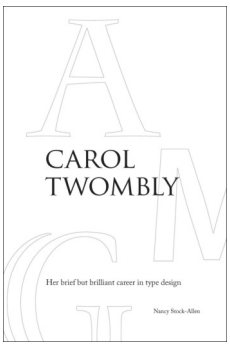
[6] Frank Mittelbach, "Laudatio for Professor Hermann Zapf", *TUGboat* **22**:1/2 (2001), 24–26, tug.org/TUGboat/tb22-1-2/tb70laud-revised.pdf

[7] Hermann Zapf, "My collaboration with Don Knuth and my font design work", *TUGboat* **22**:1/2 (2001), 26–30, tug.org/TUGboat/tb22-1-2/tb70zapf.pdf

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

Book review: *Carol Twombly: Her brief but brilliant career in type design*, by Nancy Stock-Allen

Karl Berry



Nancy Stock-Allen,
Carol Twombly: Her brief but brilliant career in type design. Oak Knoll Press, 2016, hardcover, 176pp., US$49.95, ISBN 978-1584563464.
oakknoll.com/pages/books/125344

The book begins with a welcome overview of women type designers; although women were employed in type production from at least the 19th century onward, the book reports the earliest documented design by a woman was perhaps Belladonna Kartenschrift in 1912, by Hildegard Henning for a Leipzig foundry. The careers of Gudrun Zapf von Hesse, Fiona Ross, Zuzana Licko, and many others, are all discussed. But as the title indicates, the book's subject is Carol Twombly (1959–), an extended case study of a designer's career as digital typography came of age.

Following the introduction, the book contains two main parts: Twombly's student years at the Rhode Island School of Design (RISD) and Stanford; and her career at Adobe. A third section presents samples of her released typefaces, with commentary from Twombly on each, written for the book.

In each section, Stock-Allen describes the relevant designs at length, with attention to typographic detail. Many illustrations are included, from screenshots of font design software to photographs, from marked-up proofs of draft letterforms to samples of the final type.

Of particular interest to the TEX world is the extensive section on Twombly's graduate studies at Stanford, as one of the inaugural (and, as it turned out, only) class in the Master's program in digital typography. The program was promulgated by Donald Knuth and co-chaired by Charles Bigelow, who had met Twombly as an undergraduate at RISD and invited her to Stanford.

Her time there is covered in considerable detail, including classes, conferences, the many visiting typographic luminaries, the computer science vs. artistic challenges and collaborations, and culminating in implementing Hermann Zapf's Euler design, a project widely known in the TEX world.

Following Stanford, Twombly worked at Adobe for a decade, spanning the era from the beginnings of commercial digital type to fonts becoming a commodity item. Stock-Allen concisely but informatively describes both the changing typography milieu as desktop publishing became common in the world, and Adobe's changes as a corporation.

Twombly's first designs for Adobe were the display faces Lithos, Charlemagne, and Trajan, all of which met with commercial success, followed by the Adobe Caslon Pro revival. She was then involved with the Multiple Master project at Adobe, for essentially its entire duration. Her last project for Adobe was the Chaparral Multiple Master family, a humanistic slab serif which, unusually, had design axes for changing serifs (wedge to slab), and optical size, as well as the common weight and width axes.

Stock-Allen ends the main text with comments from several noted typographers and historians on the significance and legacy of Twombly's work.

As a postlude, Twombly retired from type design (and city living) after leaving Adobe, and now pursues other artistic interests in a small town near the Sierras. As Stock-Allen observes, rhythm and writing remain an innate part of her art.

The book is superbly designed by the author and typeset in Twombly's Adobe Caslon Pro and Adobe Trajan.

◇ Karl Berry
karl (at) freefriends dot org
tug.org/books

Some of the Chaparral Multiple Master design axes.

recent works: (left) woven and beaded Shekere gourds; (right) detail from mixed media painting.

Die TEXnische Komödie 2–3/2019

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 2/2019

Graham Douglas, Wie funktionieren TEX Makros, 1 [How TEX macros work, 1]; pp. 30–33

This series has an ambitious goal: to explain how TEX macros work at the lowest level. We do not focus on simple examples but look deep into TEX itself to explain how and why TEX macros work the way they do.

Hartmud Koch, Geometrische Konstruktionen mit METAPOST [Geometric constructions using METAPOST]; pp. 34–44

In this article we look at geometrical primitives in MetaPost and how we can use them to create drawings.

Walter Entenmann, Spirale entlang eines Pfades (Coil) [Drawing spirals along a path (coil)]; pp. 45–63

This article deals with the implementation of an efficient way for graphics packages to draw a spiral along a path, connecting two dots.

Lukas Bossert, Zur Nutzung von makefile-Dateien [On the use of makefiles]; pp. 64–71

Managing larger LATEX projects with many files is sometimes a bit difficult, as usually several steps need to be executed. In this article we show how GNU make and its friends can simplify the job.

Rolf Niepraschk, Nicht zu früh und nicht zu spät: „everypar“ [Not too early and not too late: “everypar”]; pp. 72–73

This article covers an issue during the creation of an index and how it can be solved.

Rolf Niepraschk, LATEX und base64-kodierte Grafiken [LATEX and base64-coded graphics]; pp. 74–76

Most graphics used in LATEX are binary formats, containing bytes with arbitrary values between 0 and 255. To transfer these via http or smtp they are encoded so that they consist of ASCII characters only.

Herbert Voss, Generieren von BIBTEX-kompatiblen Literaturdaten [Generating BIBTEX-compatible literature database entries]; pp. 77–83

Creating entries for literature databases can take substantial effort. Especially for *Die TEXnische Komödie* we often need bibliographical data on different packages. In this article we show how a Lua script or Google can provide the necessary data.

Herbert Voss, Ausgeben der Definition von LATEX-Makros [Outputting the definition of specific LATEX macros]; pp. 84–86

`latex.ltx` is what we usually mean by saying “LaTeX” as it contains the set of basic LATEX commands. If one needs the definition of a certain command, this is the place to look.

Herbert Voss, Positionsbestimmung auf einer Textseite [Getting the exact position on a page]; pp. 87–88

Normally one does not need to know the exact current position on the current page. However, there are scenarios — e.g., when one wants to put objects relative to the current position — where the knowledge of the exact location is helpful.

Die TEXnische Komödie 3/2019

Evelyn Sarna, LATEX als TEXnisches Werkzeug für die Erstellung von Editionen [LATEX as a TEXnical tool for the creation of critical editions]; pp. 11–28

With LATEX, a critical edition that meets modern standards can be easily and professionally created. Nevertheless, in the humanities and cultural studies, critical editions are typically made with Word or Classical Text Editor. LATEX is usually not considered, although with it high-quality typographic results are possible. This article presents a selection of basics for setting up a critical edition's text in LATEX using `reledmac` and `junicode`.

Walter Entenmann, ISO-80000 konformer Mathematiksatz mit LuaLATEX [ISO-80000-conformant mathematical typesetting with LuaLATEX]; pp. 28–42

The international standard ISO-80000, as of 20 May 2019, regulates the definition and usage of SI-units and the correct writing of dimensions and units especially in mathematics and other STEM subjects. This article describes the practical implementation of the ISO standard with LuaLATEX.

Wolfgang Beinert, Umbruch [Page makeup]; pp. 42–51

In the field of typography “break” means a) the breaking of text lines, columns or pages with respect to orthographical, typographical, aesthetic and topic-related rules and points of view to typeset a book, newspaper, flyer or web page; b) text that is broken at the end of a line and continues on a new line or page; c) the balancing of all linebreaks within a piece of text.

[Received from Herbert Voß.]

***Eutypon* 40–41, October 2018**

Eutypon is the journal of the Greek TEX Friends (`http://www.eutypon.gr`).

Regretfully, this was the last issue of *Eutypon*. The following text is taken from the Editorial: "Our journal persevered for more than twenty years because we enjoyed TEXing. We also enjoyed showing, albeit indirectly, to our readers the principles of beautiful typesetting of Greek documents. However, we decided to put an end to this publication, because soliciting papers had become really hard, and because our pockets went dry. We thank you for being with us in the beautiful journey of *Eutypon* and we salute you with a virtual handshake."

Although *Eutypon* will not be published anymore, the Greek TEX Friends will use their blog `http://eutypon.gr/e-blog/` to post news and articles about TEX and Greek typography.

Sergey Beatoff, Truetypewriter PolyglOTT: Your multilingual typewriter assistant; pp. 1–10

This article is not meant to be a description of any kind of scientific research. It is merely the history of how I came to like font design, what difficulties I encountered and how I found solutions. It is about creating my font Truetypewriter PolyglOTT. I describe my long way from the idea of creating the font to its different implementations, with version 3.76 as of October 2018. In the beginning, I mistakenly thought there were several fonts simulating true typewriter printing; however, after checking out, I noticed that nearly all of them included only the Latin part. Besides, they merely accentuated the defects and inaccuracies of typewriters' text, trying to make the fonts look too realistic. In the article, I also discuss the creation of computer fonts in general. (*Article in English.*)

Linus Romer, Greek letters for the Fetamont typeface; pp. 11–14

The glyph range of the Fetamont typeface has been expanded in order to support polytonic Greek. This article describes the problems and solutions that arose during the creation process. (*Article in English.*)

Tassos Dimou, Tables of function signs and variations using tkz-tab; pp. 15–35

In this article, we present how one can use the package tkz-tab to produce beautiful tables of function signs and variations. The package tkz-tab is an extension of the drawing package Ti*k*Z, from which several commands are taken. The use of tkz-tab is explained by several introductory examples, thus avoiding the complicated commands of Ti*k*Z. (*Article in Greek with English abstract.*)

Dimitris Papazoglou, Giorgos Triantafyllakos, Giorgos D. Matthiopoulos, Axel Peemöller, Designing the visual brand of the National Library of Greece; pp. 37–46

In the autumn of 2016, the National Library of Greece announced a competition for "the design of a visual identity" in view of the transfer of the Library's collection from the old Vallianeion and Votanikos buildings in central Athens to its new facilities at the Stavros Niarchos Foundation Cultural Centre in Piraeus. In February 2017, it was announced that the winner of the design contest was a team led by Dimitris Papazoglou from Thessaloniki. In this article, the winners explain the philosophy behind the new brand of Greece's main library. (*Article in Greek with English abstract.*)

Apostolos Syropoulos, Giorgos D. Matthiopoulos, Dialogue: On the GFS Neohellenic Math font; pp. 47–50

The authors debate the æsthetics, the encoding and the use of the new font GFS Neohellenic Math. (*Article in Greek.*)

Apostolos Syropoulos, Dimitrios Filippou, TEXniques: Fonts with a bit of chemistry; pp. 51–54

In this regular column, we briefly present the design of Greek characters for the font Sans Forgetica, the new free font Cavafy Script, which is based on the handwriting of Greek poet C.P. Cavafy, and Linus Romer's attempt to convert the Malvern font from METAFONT to OpenType. The authors also present a recent improvement in the chemfig package for the creation of chemical formulæ for polymers. (*Article in Greek.*)

Dimitrios Filippou, Book presentation; pp. 55–56

The following books are presented:
(a) Dilip Datta, *LATEX in 24 Hours: A Practical Guide for Scientific Writing*, Springer, Cham, Switzerland 2017; and
(b) Vincent Lozano, *Tout ce que vous avez toujours voulu savoir sur LATEX sans jamais oser le demander*, 2e édition, Framasoft, Lyon, France 2013.
(*Article in Greek.*)

[Received from Dimitrios Filippou and Apostolos Syropoulos.]

The Treasure Chest

This is a selection of the new packages posted to CTAN (`ctan.org`) from April–October 2019, with descriptions 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 and its companions will help to make CTAN a more accessible resource to the TEX community. See also `ctan.org/topic`. Comments are welcome, as always.

◇ Karl Berry
tugboat (at) tug dot org

fonts

`almendra` in `fonts`
Almendra is a new calligraphic typeface design.

`bitter` in `fonts`
Bitter is a new slab serif design.

`forum` in `fonts`
Forum is a new design with classic proportions and multilingual support.

`garamond-libre` in `fonts`
Garamond Libre is based on Textfonts, with multilingual and OpenType support.

`librefranklin` in `fonts`
Libre Franklin is a reinterpretation of the classic 1912 Franklin Gothic design.

`imfellflowers` in `fonts`
Two flower fonts by I.M. Fell, revided by Igino Marini.

`linguisticspro` in `fonts`
Linguistics Pro is based on Utopia Nova, with two Cyrillic designs.

`logix` in `fonts`
More than 3,000 symbols, especially logic-related, complementing STIX2.

`marcellus` in `fonts`
Marcellus is a new flared-serif design, inspired by Roman inscriptions.

`poiretone` in `fonts`
PoiretOne is a new decorative geometric grotesque.

`step` in `fonts`
STEP is a Times-like font family forked from STIX and XITS.

`theanodidot` in `fonts`
TheanoDidot is a modern Greek design.

`theanomodern` in `fonts`
TheanoModern is also a modern Greek design.

`theanooldstyle` in `fonts`
TheanoOldStyle is an old-style Greek design.

graphics

`codeanatomy` in `graphics`
Support code anatomy as described in `https://introcs.cs.princeton.edu/java/13flow`.

`flowframtk` in `graphics/pgf/contrib`
Java application to create vector graphics for PGF and design frames for `flowfram`.

`matrix-skeleton` in `graphics/pgf/contrib`
Simplify working with multiple matrix nodes.

`pgfmorepages` in `graphics/pgf/contrib`
Assemble multiple pages on one physical page, extending `pgfpages`.

`pst-turtle` in `graphics/pstricks/contrib`
"Turtle" graphics (left, right, etc.) in PSTricks.

`simpleoptics` in `graphics/pgf/contrib`
Drawing lenses and mirrors for optical diagrams.

info

`apprendre-a-programmer-en-tex` in `info`
Learning to program in TEX (written in French).

language/japanese

`bxghost` in `language/japanese`
Ghost insertion for proper xkanjiskip.

macros/generic

`tokcycle` in `macros/generic`
Build tools to process tokens one at a time.

macros/latex-dev

*`latex-base-dev` in `macros/latex-dev/base`
New format to simplify testing LATEX release candidates: run `pdflatex-dev` (or similar for other engines) to help test. For details, see the LATEX news item in this issue of *TUGboat* and `latex-project.org/news`.

`latex-amsmath-dev`
`latex-graphics-dev`
`latex-tools-dev` in `macros/latex-dev/required`
The core packages accompanying `latex-base-dev`.

macros/latex/contrib

`amscdx` in `macros/latex/contrib`
Enhanced commutative diagrams.

`arraycols` in `macros/latex/contrib`
New column types for `array` and `tabular`.

`asmejour` in `macros/latex/contrib`
Template for American Society of Mechanical Engineers (ASME) journals.

`bargraph-js` in `macros/latex/contrib`
Bar graphs with Acrobat forms and JavaScript.

`centeredline` in `macros/latex/contrib`
Enhanced `\centerline`.
`csvmerge` in `macros/latex/contrib`
Merge TEX code with CSV data.
`ddphonism` in `macros/latex/contrib`
Twelve-tone music matrices, clock diagrams, etc.
`derivative` in `macros/latex/contrib`
Easy and customizable derivatives.
`ehhline` in `macros/latex/contrib`
Extend `hhline` by applying LATEX commands.
`glosmathtools` in `macros/latex/contrib`
Generate a list of mathematical symbols or notation.
`glossaries-slovene` in `macros/latex/contrib`
Slovene translation for `glossaries`.
`hu-berlin-bundle` in `macros/latex/contrib`
Classes for the Humboldt-Universität of Berlin.
`inkpaper` in `macros/latex/contrib`
Mathematical paper template for `github.com/inklatex-group/InkPaper`.
`mathcommand` in `macros/latex/contrib`
Define macros with various math mode behaviors.
`labels4easylist` in `macros/latex/contrib`
Add reference labels to `easylist` items.
`mlacls` in `macros/latex/contrib`
Class for MLA (humanities) format.
`numberpt` in `macros/latex/contrib`
Typeset counters spelled out in Portuguese.
`practicalreports` in `macros/latex/contrib`
Macros to simplify report writing.
`proof-at-the-end` in `macros/latex/contrib`
Move proofs to an appendix, and more.
`pseudo` in `macros/latex/contrib`
Straightforward pseudo-code typesetting.
`quantumarticle` in `macros/latex/contrib`
Template for the Quantum journal.
`quiz2socrative` in `macros/latex/contrib`
Prepare questions for Socratic quizzes.
`scontents` in `macros/latex/contrib`
Store and reuse code sequences.
`spacingtricks` in `macros/latex/contrib`
New and updated macros for improved spacing.
`subtext` in `macros/latex/contrib`
Easy `\text`-style subscripts in math mode.
`thuaslogos` in `macros/latex/contrib`
Logos of The Hague University of Applied Sciences (THUAS).
`tuda-ci` in `macros/latex/contrib`
Templates for the Technische Universität Darmstadt.
`unam-thesis` in `macros/latex/contrib`
Support for National Autonomous University of Mexico (UNAM) dissertations.
* `unicode-alphabets` in `macros/latex/contrib`
Macros for characters from various Private Use Area (PUA) character sets in LATEX (CYFI, MUFI, SIL, TITUS, UNZ).
`unifith` in `macros/latex/contrib`
Theses at the University of Florence in Italy.
`unizgklasa` in `macros/latex/contrib`
Theses at the Faculty of Graphic Arts in Zagreb.
`vtable` in `macros/latex/contrib`
Vertical alignment of table cells.
`yazd-thesis` in `macros/latex/contrib`
Theses at Yazd University in Iran.

macros/luatex/latex

`addliga` in `macros/luatex/latex`
Access basic f-ligatures in TrueType fonts lacking a `liga` table.
`hmtrump` in `macros/luatex/latex`
Typeset playing cards.
`pdfarticle` in `macros/luatex/latex`
Class for PDF publications with LuaLATEX.

macros/xetex/latex

`nanicolle` in `macros/xetex/latex`
Herbarium specimen labels in Chinese.
`quran-ur` in `macros/xetex/latex`
Urdu translations for the `quran` package.

support

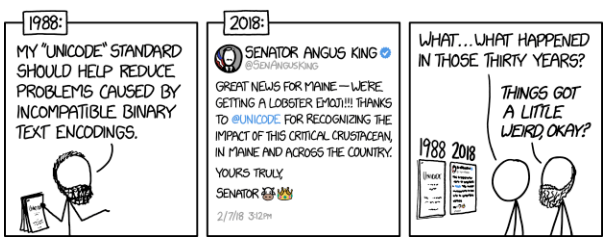
`zblbuild` in `support`
Shell/GUI script to help select a BIBLATEX style and options.

systems

`pdftex-djgpp` in `systems/msdos/djgpp-contrib`
Binary for `pdftex` for the `msdos-djgpp` distribution.

web

`clojure-pamphlet` in `web`
Literate programming based on Clojure pamphlets.



Comic by Randall Munroe (`https://xkcd.com`).

TEX Development Fund 2014–2019 report

TEX Development Fund committee

These projects (listed by reference number) have been supported in recent years by the TEX Development Fund. For application and donation information, the complete list of supported projects, and more, please see: https://tug.org/tc/devfund

32. HarfTEX engine based on LuaTEX

Applicant: Khaled Hosny, Egypt.
https://github.com/khaledhosny/harftex
Amount: US$1000; acceptance date: 1 May 2019; completed 9 September 2019.

HarfTEX is a TEX engine based on LuaTEX, extending it with HarfBuzz, ICU and possibly other libraries for Unicode text layout and modern fonts support.

The engine will be extended to offer more libraries, and to fix some of the limitations faced during the previous stage of development.

This project has been adapted into the LuaHBTEX engine, now maintained as part of the LuaTEX sources by Luigi Scarso.

31. HarfBuzz access from LuaTEX

Applicant: Khaled Hosny, Egypt.
Amount: US$2000; acceptance date: 1 November 2018; completed 17 April 2019.

Provide a set of Lua modules that bridge LuaTEX and HarfBuzz (possibly/eventually also ICU, FreeType, and FontConfig), and bundle these modules with LuaTEX to extend its functionality with the ability to typeset more world languages and scripts. Also provide Lua code that integrate these modules with LuaTEX callbacks.

30. Documentation in Persian of the core TEX system

Applicant: Vafa Khalighi, Australia.
Amount: US$1000; acceptance date: 10 October 2018.

Comprehensive documentation in Persian of TEX, Metafont, and Computer Modern, including WEB programming, Pascal programming (as used in *.web), and other material.

29. Light (LA)TEX Make

Applicant: Takuto Asakura, Japan.
https://github.com/github.com/wtsnjp/llmk
Amount: US$1000; acceptance date: 25 July 2018; completed 30 August 2019.

The procedure of building is essential for LATEX documents to get the correct output, as the creator intends. The procedure of building includes the information like which TEX engine to use, which outer programs to use, what options and arguments and/or which configuration files should be given for each program, and the processing order of the programs.

The proposed features of the llmk program are as follows (all subject to change with more experience):

- Working solely with Lua on LuaTEX (texlua). Neither external programs nor any Lua libraries from third parties are required.
- Using TOML to declare the settings. TOML (https://github.com/toml-lang/toml) is a minimal configuration format similar to JSON and YAML. I selected TOML because it is human readable and its specification is relatively simple, making it easy to write a parser from scratch.
- No complicated nesting of configuration. llmk allows users to write configuration (in TOML format) in the source of LATEX documents or the special file (named llmk.toml). Since the procedure of building is essential and independent for each project, llmk does not load the configuration files recursively.
- Modern default settings. llmk will try to make a modern de facto standard. For instance, if an user doesn't specify any TEX engine to use, llmk will use LuaTEX to compile the source.

Current status and project goals:

- The most basic functionality of llmk is already implemented (see the README at the project page for the details). In the year, I'm going to enhance the program including:
- full support for TOML specification (currently, only partially supported)
- supporting some types of magic comments (e.g., those of TEXworks)
- full documentation (currently, only README)
- other convenient features (e.g., quiet mode and an option to support specifying an output directory)

28. MetaPost updates

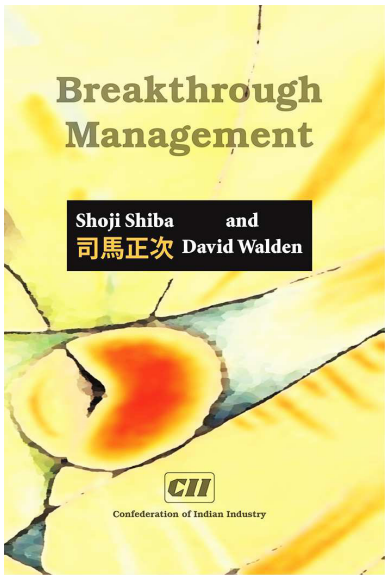
Applicant: Luigi Scarso, Italy.
Amount: US$1000; acceptance date: 28 December 2015; completed 24 May 2017.

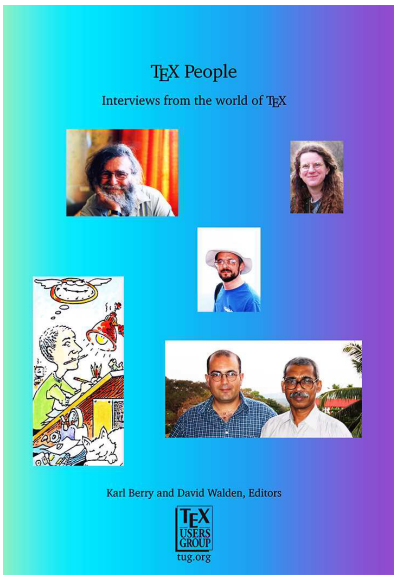
Bug fixes and maintenance for MetaPost path resolution, binary, decimal, and double number systems. Also, more efficient integration with mplib by excluding the libraries needed only in LuaTEX. A report from May 2017 is available at https://tug.org/devfund/documents/2017-05-scarso.pdf.

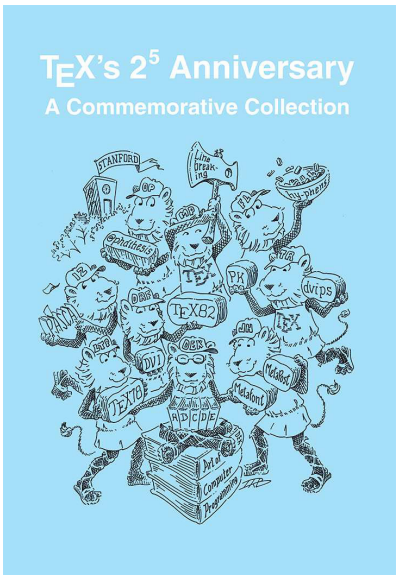
27. Libertine OpenType math fonts

Applicant: Khaled Hosny, Egypt.
https://github.com/khaledhosny/libertinus
Amount: US$2000; acceptance date: 24 December 2014; completed 1 February 2016.

Building an OpenType math companion for Linux Libertine and Linux Biolinum fonts (ultimately named Libertinus), and fixing bugs in those fonts. Also coordinating with the authors of Linux Libertine (LA)TEX support files to adapt the new and fixed fonts (ultimately they did not respond, so upstream Linux Libertine is unmaintained as far as we can tell). The package has been released to CTAN: https://ctan.org/pkg/libertinus-otf. ◇







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.

Aicart Martinez, Mercè
Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) ono.com`
Web: `http://www.edilatex.com`
We provide, at reasonable low cost, LaTeX or TeX page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

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.

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, C, C++ 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.

(continued)

Veytsman, Boris (cont'd)

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
6504681393
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.

Calendar

2019

Oct 19	DANTE 2019 Herbsttagung and 61st meeting, Kirchheim unter Teck, Germany. www.dante.de/veranstaltungen/herbst2019
Oct 25	Award Ceremony: The Updike Prize for Student Type Design, Providence Public Library, Providence, Rhode Island. www.provlib.org/updikeprize
Oct 26	GuIT Meeting 2019, XVI Annual Conference, Turin, Italy. www.guitex.org/home/en/meeting

2020

Feb 28 – Mar 1	Typography Day 2020, "Typographic Dialogues: Local-Global". Beirut, Lebanon. www.typoday.in
Mar 25 – 27	DANTE 2020 Frühjahrstagung and 62nd meeting, Lübeck, Germany. www.dante.de/veranstaltungen
Apr 24 – 25	Before & Beyond Typography: Text in Global & Multimodal Perspective, Stanford University, Stanford, California. www.eventbrite.com/e/before-beyond-typography-text-in-global-multimodal-perspective-tickets-69068930029
Apr 29 – May 3	BachoTEX 2020, 28th BachoTEX Conference, Bachotek, Poland. www.gust.org.pl/bachotex
Jun 4 – 6	Markup UK 2020. A conference about XML and other markup technologies, King's College, London. markupuk.org
Jul 1 – 3	Eighteenth International Conference on New Directions in the Humanities, "Transcultural Humanities in a Global World", Ca' Foscari University of Venice, Venice, Italy. thehumanities.com/2020-conference
Jul 19 – 23	SIGGRAPH 2020, "Think beyond", Washington, DC. s2020.siggraph.org
Jul 22 – 24	Digital Humanities 2020, Alliance of Digital Humanities Organizations, Carleton University and the University of Ottawa, Ottawa, Canada. adho.org/conference

TUG 2020 Rochester Institute of Technology, Rochester, New York

Jul	The 41st annual meeting of the TEX Users Group. tug.org/tug2020

Sep 6 – 12	14th International ConTEXt Meeting, Prague-Sibřina, Czech Republic. meeting.contextgarden.net/2019

Status as of 15 October 2019

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.

TUGBOAT Volume 40 (2019), No. 3

Introductory

Intermediate

Intermediate Plus

Advanced

Reports and notices