TUGboat

Volume 43, Number 3 / 2022

TEX Users Group

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

Individual memberships

2023 dues for individual members are as follows:

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

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

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

Journal subscriptions

TUGboat subscriptions (non-voting) are available to libraries and other organizations or individuals for whom memberships are either not appropriate or desired. Subscriptions are delivered on a calendar year basis. The subscription rate for 2023 is $115.

Institutional memberships

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

Trademarks

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

[printing date: November 2022]

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`

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

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

For a printer is quintessentially a "worker intellectual or an intellectual worker", the very ideal of that human type who would become the pivot of socialism: "the conscious proletarian".

Régis Debray
"Socialism: A Life-Cycle",
New Left Review **46** (2007)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 43, NUMBER 3, 2022
PORTLAND, OREGON, U.S.A.

TUGboat editorial information

This regular issue (Vol. 43, No. 3) is the last issue of the 2022 volume year. The deadline for the first issue of next year is March 24, 2023. Contributions are requested.

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

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

TUGboat advertising

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

Submitting items for publication

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

The *TUGboat* style files, for use with `plain` TeX and LaTeX, are available from CTAN and the *TUGboat* web site, and are included in TeX distributions. We also accept submissions using ConTeXt. For deadlines, templates, author tips, and more, see `tug.org/TUGboat`.

Effective with the 2005 volume year, submission of a new manuscript implies permission to publish the article, if accepted, on the *TUGboat* web site, as well as in print. 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,* `ams.org`

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

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

Center for Computing Sciences, *Bowie*, *Maryland*

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

CTAN, `ctan.org`

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

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

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

L3Harris, *Melbourne*, *Florida,* `l3harris.com`

LaTeX Project, `latex-project.org`

MacTeX, `tug.org/mactex`

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

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

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

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

Overleaf, *London*, *UK,* `overleaf.com`

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

TeXFolio, *Trivandrum*, *India,* `texfolio.org`

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

University of Ontario, Institute of Technology, *Oshawa*, *Ontario*, *Canada,* `ontariotechu.ca`

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

VTeX UAB, *Vilnius*, *Lithuania,* `vtex.lt`

From the president

Boris Veytsman

Something there is that doesn't love a wall

Robert Frost

Attentive readers of *TUGboat* might have noticed a change over the last few issues: articles are now accompanied by unique strings of letters and numbers, their Digital Object Identifiers, or DOIs for short. This would be impossible without hard work done by the *TUGboat* team, and especially Karl Berry, who made many updates to our styles and scripts. Some of these updates may help other publishers. They are released to CTAN and CPAN.

This is an important change. Science and technology are collective endeavors rather than the heroic product of inspiration of singular geniuses. We all build the edifice started by our predecessors, and our papers or programs contain acknowledgments of them. Historically formatting of citations is defined by the "house style" of the publisher, and it took much manual effort to find the canonical citation of the referenced work (my colleagues and I published some research about this on these pages). Exact citation data are useful for the funders (is my money doing good work?), for the practitioners (is my work visible?) and for those studying the science of science (how is knowledge generated and spread?). However, it is very difficult to work with the data without associating a unique identifier with each entity — and this is what DOIs are for. Each paper gets a unique DOI, and it makes it much easier to study the citation relationships between the different papers — especially when authors' own unique identifiers (called ORCIDs) are used.

Assigning DOIs requires some important decisions. The first is, what should get a DOI, and what should not? Every technical paper, sure. Editorials and columns like this one, yes. What about tables of contents? Indices? Advertisements? We made the decision not to assign DOIs to these ephemera, which have no assigned author. This decision might be controversial, but any choice can be criticized. Note that Crossref, the authority controlling the registration of scientific DOIs, elegantly sidestepped this choice: member organizations like TUG must make it themselves for their publications. The entities that get DOIs are not just papers: any "digital object" can obtain one, including programs, datasets, etc. This is a smart decision: if DOIs were assigned to papers only, then we would have a hard question: what is a paper? In a recent work my colleagues and I catalogued the use of scientific software, and found that the question, what is software?, is rather difficult to answer: the differences between software packages, algorithms, Web services, etc., are not evident. Are Google spreadsheets software or a Web service? What about LAPACK with its multiple implementations: is it an algorithm or software?

Another hard question is whether the entity to be given a DOI is one, or many. We do not assign DOIs to separate sections of the same paper. What if the paper is split into parts and published in several consecutive issues? Yet another question is raised when a paper is updated. *TUGboat*, being a physical journal (even if many readers get it in the digital form), has a reference version of each paper. Not so for some other publishers like preprint servers, which allow posting new version of the same paper. Do these versions deserve a separate DOI each? What if between updates the title or the abstract changed? Or if the team of authors changed (strangely, this happens too)?

These questions remind me of the famous problem of the ship of Theseus in philosophy. If we change a rotten plank on this ship, it is still the same ship. What if we change two planks? Three? What if in several centuries we have changed all planks — is it still the ship of Theseus?

There are several ways to treat this paradox. My favorite one is to recognize that the world is infinitely complex, and our labels, like "scientific paper", "ship", etc., are just approximations of it. These labels are useful, but they are not the reality. Thus they are bound to break in some cases.

A similar problem occurs in a field better known to TeXnicians. As many of us know, TeX itself is a program that puts letters on paper. Formats like LaTeX or ConTeXt consider the structure of the text, so `\section{Introduction}` in LaTeX means that (a) we start a new unit of text, called "Introduction", and (b) tell TeX to typeset it in a certain way (bold, or in small caps, or maybe in larger size, etc.), possibly starting the next paragraph without indentation, update the table of contents, etc. Ideally a document in these systems should not have any visual TeX commands, as they relate to how the text is shown rather than what is it.

However, as anybody using these systems knows, more often than not this is not true. Even if the authors do not abuse visual commands, adding unnecessary spaces or decorations, they often need some presentation effects that are simply not expressible by the high-level commands. It looks like our ideas of what constitutes text structure are as approximate as any other ideas — like the ideas of "scientific paper"

doi.org/10.47397/tb/43-3/tb135pres

or "ship". It is not a coincidence that our colleagues who represent text in XML format use "extensibility" in the name of their approach (XML stands for eXtensible Markup Language). They implicitly admit that any specification is not complete and may need to be extended.

Perhaps nowhere is this problem so evident as in the language of mathematics. As editors of mathematics journals know too well, mathematicians are fond of inventing new notation, stretching the limits of typesetters and TeXnicians. I think it is not a coincidence or another manifestation of the peculiar nature of mathematicians themselves. Rather, the existing notation reflects our existing knowledge, and if we actively work on extending the knowledge, we necessarily need to extend our notation.

As an aside, this makes Knuth's structure of mathematical formulae, with eight classes of mathematical objects, even more surprising since it has endured several decades of heavy use. In general, TeX mathematical typesetting is still very close to that of the original version. A notable exception is the `\middle` primitive added by ε-TeX to `\left` and `\right`. LuaTeX adds a number of other primitives (and development continues there). It will be interesting to see whether they will be adopted by other engines in the future.

At the end of the day these questions, from the ship of Theseus to DOI assignment to mathematical notation, are questions of classification. Jorge Luis Borges touches this problem in his famous essay *The analytical language of John Wilkins*, where his protagonist invents a meaningful language, which is also "a secret encyclopedia":

> He divided the universe in forty categories or classes, these being further subdivided into differences, which was then subdivided into species. He assigned to each class a monosyllable of two letters; to each difference, a consonant; to each species, a vowel. For example: *de*, which means an element; *deb*, the first of the elements, fire; *deba*, a part of the element fire, a flame. In a similar language invented by Letellier (1850) *a* means animal; *ab*, mammal; *abo*, carnivore; *aboj*, feline; *aboje*, cat; *abi*, herbivore; *abiv*, horse; etc. In the language of Bonifacio Sotos Ochando (1845) *imaba* means building; *imaca*, harem; *imafe*, hospital; *imafo*, pesthouse; *imari*, house; *imaru*, country house; *imedo*, column; *imede*, pillar; *imego*, floor; *imela*, ceiling; *imogo*, window; *bire*, bookbinder; *birer*, bookbinding. (This last list belongs to a book printed in Buenos Aires in 1886, the *Curso de Lengua Universal*, by Dr. Pedro Mata.) *[Translated from Spanish by Lilia Graciela Vázquez; edited by Jan Frederik Solem with assistance from Bjørn Are Davidsen and Rolf Andersen.]*

This long quotation reveals the major problem with any classification: it is always "a secret encyclopedia". However, an encyclopedia requires absolute knowledge. If our understanding of nature is not complete, the classification cannot be right. John Wilkins in the 17th century considered fire a primary element, and his language reflected it. We now consider fire a complex physico-chemical process. Thus Wilkins' language is inadequate for us. A similar problem is with Linnaeus' nomenclature in botany and zoology. It reflects our knowledge of the relationship between the species. If the knowledge is updated, for example, due to molecular genetics methods, the nomenclature becomes "wrong". Since our knowledge is never absolute, our classification is never right.

Yet another way to express this is that stable mathematical notation is possible only for *dead* mathematics, a dry school subject rather than living and developing field. The latter requires constant changing of the notation.

One can compare our efforts in classification to wall building. We constantly build walls around the things we know, delineating our knowledge. However, as Robert Frost noted (*Mending Wall*, 1914),

> Something there is that doesn't love a wall,
> That sends the frozen-ground-swell under it,
> And spills the upper boulders in the sun,
> And makes gaps even two can pass abreast.

Does this mean that our efforts are futile? No. This wall building is our effort of understanding the infinitely complex world, creating better and better models for it — while always remaining imperfect.

Some prefer to think of this work as building roads rather than building walls. In this case another metaphor is apt, this one by Ukrainian poet Ivan Franko (*The Stonecutters*, 1878): *we are but stone cutters for the road of progress.* It is interesting that both Frost and Franko think about stone as the hard material we are working on. For a long time stone was used to make inscriptions for the future — something intimately close to us TeXers.

Like Frost's protagonist and his neighbor, "we wear our fingers rough" creating imperfect inscriptions with our imperfect tools. We are bound to do this till the end times.

◇ Boris Veytsman
`president (at) tug dot org`

Editorial comments

Barbara Beeton

New from Don Knuth

The "next" installment of *The Art of Computer Programming* (*TAOCP*), volume 4B, *Combinatorial Algorithms*, has been published this month (October 2022). A discount is available on all orders made directly from the publisher; see the TUG bookstore[1] for more information.

More memories of Dave Walden

Among his other pursuits, Dave was active on the editorial board of the *IEEE Annals of the History of Computing.* The story of all his exploits, "David Corydon Walden's Five Careers", has been told in that journal by Alexander A. McKenzie, and is open to be read by anyone. Read it[2] — it's fascinating.

Dave's last article for the journal, co-authored with McKenzie and W. Ben Barker, "Seeking high IMP reliability in maintenance of the 1970s ARPAnet", appeared in the previous issue[3] behind a paywall, but should be readable by IEEE members.

Update on arXiv and HTML

A note in my column in *TUGboat* **43**:1 announced ar5iv, an HTML counterpart to arXiv. Shamsi Beers Brinn has provided an update:

> `arXiv.org` is working on improving the accessibility of research papers posted on the site. We want to get the word out about this effort as we work on building support and securing funding. A paper about this, and blog posts summarizing it, will be available in November at `blog.arxiv.org` and will also be announced on the usual TeX-related mailing lists. One proposed next step of note that will be discussed: arXiv's offering of HTML alongside existing PDF and TeX formats.
>
> ar5iv is the public name for the LaTeXML project, a long-standing independent research effort run by Bruce Miller and Deyan Ginev from NIST. ar5iv is also under the "arXiv Labs" umbrella, which is a structure for us to work with outside projects that offer significant value to arXiv's users.
>
> For our accessibility work, we are actively investigating the best option for generating HTML from TeX. LaTeXML is one of the front runners so it is possible that we will have a closer relationship with them in the future.

[1] `tug.org/books/`
[2] `www.computer.org/csdl/magazine/an/2022/03/09842292/1FlM1ABQDsI`
[3] `www.computer.org/csdl/magazine/an/2022/02`

Chuck Bigelow on fonts

The October 2021 issue of *Hour Detroit* carried an interview with Chuck Bigelow on the subject "What Makes a Great Font".[4] (Chuck grew up in Michigan, in case you're wondering why this is in a Detroit magazine.) In the interview, Chuck shares high points of his growing up, how he became interested in fonts, how he and Kris Holmes came to create Lucida, and thoughts on Lucida's acceptance and future.

This interview, in turn, was picked up by the European blog *typeroom*[5] and transmogrified by some cuts, with the addition of more history and copious illustrations of Lucida, as well as a link to an oral history interview with Chuck from the Computer History Museum.

A remarkable collection of printing blocks: the Tripitaka Koreana

I have often opined that the most durable medium in which to record knowledge is clay blocks, which endure fires, becoming even more durable. But clay blocks are one-off, not meant for reproduction. For that purpose, before movable type, wood blocks were used.

The Tripitaka Koreana is a collection of over 80,000 wood printing blocks created in the 13th century that contain the oldest intact version of the Buddhist canon in the Hanja script. An exposition of this collection with excellent illustrations appears online on Twitter.[6]

Food and fonts

An important effort in the marketing of a font is to show potential buyers how it might be used for best effect. A traditional method is to present notable and recognizable quotes, often chosen for their subject matter as well as the appearance of particularly distinctive characters, along with a "showing" of the complete alphabet and associated digits and punctuation.

[4] `hourdetroit.com/art-topics/lucida-designer-chuck-bigelow-on-what-makes-a-great-font`
[5] `typeroom.eu/lucida-type-designer-charles-bigelow-what-makes-a-great-font`
[6] `twitter.com/incunabula/status/1574546784365445136`
Given current questions about the stability of Twitter, it's worth observing that many other sources of textual information can be found readily via a Web search, but mostly without the wealth of pictures.

The type foundry Commercial Type has chosen a different approach. They commissioned articles about food, and used those texts as a platform to show off their fonts. The result can be viewed at `foodissue.commercialtype.com`. More usual showings of the fonts are accessed by links in the margin of the article text.

Interlude: Overfull `\hboxes`

A link (`twitter.com/jamesdoesastro/status/1541792788475420672`) in the `tex.stackexchange` chat led me to this delightful bit of LaTeX-related doggerel.

LaTeX is strange,
It drives me to madness,
My overfull hbox
Of 10000 badness

But that "hbox of 10000 badness" would be *underfull*. (Overfull boxes are reported differently.) One gets rattled. How often has this happened to you?

Thinking this worthy of wider circulation, but being unwilling to publish it without permission, I sent a wild plea to the author's adviser. In due course, I received a message from the author, James Garland, at the time newly graduated from Haverford College, granting the desired permission. Thanks, James.

Other readers of the Twitter post added more verses. You'll have to read those for yourself.

Errata, *TUGboat* 43:1

- Page 79, *Zpravodaj*: The author of "TEX in a nutshell" is Petr Olšak, not Petr Sojka. (Our apologies to them both.)
- Cover 3:

  The title of the article by Jacques André et al. (page 7) should be "The last decade at GUTenberg", not "Year 2020 at GUTenberg".

  The article beginning on page 10, "Markdown 2.15.0: What's new?" has four authors: Vit Novotný, Dominik Rehák, Michal Hoftich, and Tereza Vrabcová.

  Information on the articles themselves and in the contents list on cover 4 is correct, and the cover 3 web page has been corrected.

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

Regarding TUG and UK-TUG

Jonathan Fine

1 Background

About 30 years ago I joined TUG and UK-TUG, the TEX users group in the United Kingdom. Sadly, UK-TUG is now dissolved. Former members include Sebastian Rahtz (founder of TEX Live), Jonathan Kew (author of XeTEX and TEXworks) and Robin Fairbairns (editor of TEX FAQ and CTAN maintainer). Its 80 former members are now dispersed, without an effective online forum.

At this year's TEX conference I gave a personal history of UK-TUG. I am most grateful to organisers, speakers and audience for these online conferences. They are a shining star in TUG's recent history.

This article is about the relationship between TUG and UK-TUG during the process that ended with the dissolution of UK-TUG. It was not pleasant and shows weakness of character in several of us, including myself. Don Knuth teaches us to learn from our errors and failures.

My starting point is this year's TUG AGM. Our Secretary Klaus Höppner wrote in his report: *TUG was not involved [in] and has no knowledge about discussion within UK-TUG about how to distribute the money [arising from the dissolution of UK-TUG].*

2 Arthur Rosendahl, TUG and UK-TUG

The relevance and truth of Klaus's statement depends on what it means. It is true that TUG Vice President Arthur Rosendahl (formerly Reutenauer) has a deep knowledge of this discussion.

As a Committee Member Arthur wrote to the UK-TUG members:

> Our organisation serves no useful purpose. The benefits of membership are already available elsewhere:
>
> - technical: Stack Exchange, Learn LaTeX, ...
> - social+technical: Facebook (UK TEX Users Forum).
> - TEX Live DVDs: existing channels will continue.
>
> It has assets that it cannot usefully spend to support TEX in the United Kingdom, and is almost entirely disconnected from the UK's TEX community. It has failed to attract new members. Its committee lacks the energy that would be necessary for a renewal.

> Its main agenda item for the last few years has been disbanding. How might any prolongation be of any benefit?
>
> We urge you to vote for motion DS1 by David Saunders, which offers the only way out of the current impasse.
>
> Signed: all but one current committee members: Arthur Reutenauer, Chris Rowley, Jay Hammond, Jonathan Webley, Simon Dales.

3 Lessons for TUG

Note well that the signatories state that the Committee, of which they are members, *lacks the energy that would be necessary for a renewal*. They also state that dissolution is *the only way out of the current impasse*. There are questions here for TUG, most important of which is: How did UK-TUG get to this situation? Will TUG share the same fate?

These are important questions. I feel a duty to the community to share my experience, hence this article. Although it has flaws, it is the best I can do.

4 UK-TUG negotiations with TUG

Arthur said he negotiated with TUG on behalf of UK-TUG. He wrote to UK-TUG members:

> I can also say that UK-TUG has essentially lost all credibility with our sister organisations, as is demonstrated by the fact that I couldn't negotiate better conditions for TUG to receive a donation from UK-TUG, and that I didn't get any official answer from DANTE when I contacted them.

In this context 'conditions' meant that the donation must go to TUG's General Fund. Such funds can be applied to office expenses, which are just over 70% of TUG's total income.

5 The Development Fund

I expressed concern at this and stated my intent to contact the TUG Board for clarification. The next day Arthur wrote *Any donation to TUG can, of course, be directed according to the many categories listed on https://tug.org/donate.html*.

I understood this reversal to mean that any dissolution donation would go to the TUG Development Fund or the like. Such donations are ring-fenced for that purpose. Such is, I believe, required by the UK-TUG Constitution.

I warmly welcomed Arthur's reversal of his previous statement. I fully expected the donation to go to the Development Fund. I now bitterly regret not then pressing the point and obtaining assurances.

6 The donation to DANTE

By the way, Arthur also wrote: *DANTE has finally and officially agreed to receiving a donation from UK-TUG upon dissolution, if it happens.*

The TUG Secretary is also a Board member of DANTE. It is likely that he knew of this offer made on behalf of UK-TUG by a person who is also Vice President of TUG.

7 What happened, my TUG AGM motion

That was in October 2021. In April this year, after the payment to TUG has been made, I was told it was sent to the General Fund (and so could be applied to the 70% of income Office Expenses).

To change this, for this year's TUG AGM I submitted the motion:

> This TUG General Meeting asks that the dissolution donation made by UK-TUG to TUG's General Fund be transferred to TUG's TeX Development Fund. For clarity, this motion is advisory and is not binding on the TUG Board.

together with a supporting statement. I took care to avoid the difficult issues I've previously mentioned.

8 Norbert Preining's contribution

I also sent the motion and supporting statement to the TUG members list. In the resulting discussion Board member Norbert Preining wrote to that list:

> I consider this ill will versus the very association you are a member of, which in my opinion can carry consequences for you.
>
> If it would be my call — for example in other associations where I am board member — your continuous passive aggressive behaviour, unfounded insinuations, and spreading of fud would have been a reason for taking the membership from you.

This message was directed at me. I read it as a threat to have me expelled as a member of TUG if I continued my present course of action. I am ashamed to say that this threat was successful.

9 At the TUG AGM

We have now returned to the starting point. Because of Norbert's message, and Klaus not circulating my motion to the members, at the AGM I found that I lacked the courage to insist that my motion be discussed and voted upon. I feared that if I did so then I would be expelled or otherwise punished.

10 Summary

Arthur had a dual role. He was a Board/Committee member of two organisations. In the one he promoted a motion that gave a substantial donation to the other.

Klaus did not bring my motion to the attention of members, and made a statement that obscured Arthur's dual role.

Norbert in so many words threatened me with expulsion from TUG.

11 Looking forward

Although UK-TUG has slipped away, I am still a member of TUG. Sometimes loyalty to our goals and purpose requires dissent from the majority and opposition to the Board. To understand better me and my background, and the sources of my strength of feeling, please watch my talk, *A personal history of UK-TUG*: `https://youtu.be/2l5yVb97I7A`.

In that talk I took as my theme John Donne's poem *For whom the bell tolls*. The demise of UK-TUG diminishes the whole TeX community.

To rephrase Donne: Do not ask for whom the UK-TUG funeral bell tolls, dear TUG member. It tolls for our organisation.

I fear for the worst. I hope for the best. I hope that I will live long enough to see TeX (including related software) and its users flourishing in 2042, twenty years from today and 50 years after my first encounters with TUG and UK-TUG.

When we sincerely work together for goals that contribute to human welfare and happiness, interpersonal difficulties become less important and can dissolve. This is how TUG was when and before I joined the organisation.

12 Added in proof

Editorial constraints prevent more from this email I sent to Arthur being published in this issue of *TUGboat*. Therefore I will publish the whole message on my website (see URL below), once this issue is published. I am willing to discuss matters relating to TUG and UK-TUG with those who are interested, either privately or in public as is wished and appropriate. From time to time I will write on these matters on my website.

◇ Jonathan Fine
Milton Keynes, UK
`jfine2358@gmail.com`
`https://jfine2358.github.io`

Rebuttal

Arthur Rosendahl

Τί δὲ βλέπεις τὸ κάρφος τὸ ἐν τῷ ὀφθαλμῷ τοῦ ἀδελφοῦ σου, τὴν δὲ ἐν τῷ σῷ ὀφθαλμῷ δοκὸν οὐ κατανοεῖς;
Matthew 7:3

Jonathan can't have it both ways: either I didn't use my position as a link between TUG and UK-TUG efficiently enough, or I abused it. Obviously, I reject both assertions. I tried to help in a difficult debate, I had a fine line to walk, and it's always easy to find aspects to criticise one year after the fact. I never claimed to be perfect. Most important, I was extremely clear about what Jonathan calls my "dual role"; which he should know, since he quotes from an email by me where I explained exactly that. He conveniently omits the parts where I said that I was aware that I was formally in a conflict of interest, and how I planned to mitigate it.

Perhaps Jonathan would like to be reminded of his own conduct and quoted out of context too; but the predictable outcome of his actions is probably punishment enough. His nauseating insistence that UK-TUG never, never, never should be dissolved gained him no supporters, and quite a few opponents. In a way, I should be grateful.

In the end, UK-TUG was dissolved by the clearly expressed will of its members, who passed a dissolution motion 26 to 7. Two other motions, proposed by Jonathan and mutually contradictory, received 0 votes.

◇ Arthur Rosendahl
`arthur (at) rosendahl dot io`

doi.org/10.47397/tb/43-3/tb135rosendahl-uktug

The TEX Hour

Jonathan Fine

1 History

I write as the founder and host of *The TEX Hour*, a weekly online video meeting devoted to TEX and related matters, especially access to STEM documents. It also covers related computer technology.

About two years ago I started the TEX Hour as a response to both the possible dissolution of UK-TUG and also the Covid pandemic. Since then we've had almost 100 meetings, most of which have been recorded and published.

The website `https://texhour.github.io` has up-to-date information about future meetings and links to videos of previous meetings. The TEX Hour usually meets 6:30 to 7:30pm on Thursdays, BST (i.e. UK) time. All are welcome.

2 Gradual growth: TEX in STEM

The TEX Hour now has sturdy roots and is gradually growing. Last month we had a Special TEX Hour, which nicely shows how the weekly meetings fit into the annual cycle of TEX conferences.

This special TEX Hour was on *Rethinking TEX in STEM*. It was two hours. The first hour was talks from speakers from recent TEX Conferences, namely Peter Williams, Martin Ruckert and Dennis Müller. The second hour was discussion.

We are fortunate that TEXnical people at the arXiv attended this TEX Hour. The resulting discussion showed very strongly how the weekly TEX Hour cycle complements and adds value to the presentations at the annual TEX conference. This is best experienced by visiting the website for 29 September 2022 TEX Hour.

3 LATEX and Accessibility

There will be a similar special TEX Hour, devoted to LATEX and Accessibility, on Thursday 1 December at 6:30pm BST (i.e. UK) time. We will be fortunate to have Jonathan Godfrey, a blind lecturer in Statistics, as a special guest.

As Jonathan lives in Auckland, New Zealand for him this TEX Hour starts at 7:30am on Friday 2 December! More details will appear on the website.

4 Open source video meeting platform

A recent TEX Hour announcement prompted some discussion on TEX-fora hosted by TUG, about using an open-source video meeting platform. I moved this discussion off-list. I'm pleased that Paulo Ney de Souza joined that discussion. Paulo is a driving force behind the TEX conference organised by TUG.

Paulo and I formed a rough agreement that the most important issues for using an open source video meeting platform were:

1. Availability of clients on all platforms, including Chromebooks, Android and iOS.
2. Stability, including the use of servers at the Calyx Institute.
3. Bandwidth to deal with meetings of 10, 100 and 1K participants.
4. Availability of features necessary for conducting small meetings classes, office hours, small and large conferences.
5. Recording and YouTube streaming.
6. Accessibility, particularly to blind and visually disabled.
7. Ability to handle mathematical content, either directly or via a plug-in.

I welcome using the TEX Hour as a trial for use of an open-source video meeting platform. I'm personally very interested in the online sharing of mathematical content.

Surely we can in 2022 do better than using screen sharing to communicate a bitmap video of a Beamer presentation. Such improvements would provide a strong reason for preferring an open-source platform.

5 Our Zoom license

Meanwhile, we continue to use Zoom. I thank River Valley Technologies for funding our license for the next 12 months. I'm happy to use this license to host other TEX-related online meetings. I'm also very willing to open up the TEX Hour for any TEX-related topics (loosely understood) that you'd like to discuss or present on. All are welcome, especially new users and other beginners. They often ask the best questions!

6 Conclusion

To summarize. For up-to-date information about past and future meetings visit the TEX Hour website. Meetings are usually every Thursday, 6:30 to 7:30pm BST (i.e. UK) time.

◇ Jonathan Fine
Milton Keynes, UK
`jfine2358@gmail.com`
`https://texhour.github.io`

doi.org/10.47397/tb/43-3/tb135fine-texhour

DANTE project funding of TEX servers

Stefan Kottwitz

Abstract

For several years, DANTE e.V. has been supporting the operation of two physical servers that provide numerous TEX and LATEX related services. This article is a report on what is provided and how the services came about. This article originally appeared in *Die TEXnische Komödie* 1/2022, and was translated to English by the author.

1 Hardware and hosting

The DANTE-supported servers are located in two data centers of `Hetzner.com`. They run a Debian Linux with virtualization by Xen. The services mentioned in the following sections run on a number of virtual machines for clean separation and maintainability. The servers back up each other on the one hand and, on the other hand, an external backup area is also used. In the case of upgrades or migrations of virtual servers, the respective other server is used for the reinstallation.

The basic specifications of the two servers:

- Intel Xeon E5-1650V3 six-core, 2x4 terabyte hard drives, 256 gigabytes of RAM
- Intel i7-920 quad-core, 2x2 terabyte hard drives, 48 gigabytes of RAM

Both have a 1 Gbps network connection and unlimited traffic volume, as well as an off-site backup area. This hardware, with all services in professional data center operation, currently costs €104 and €59 per month, plus small fees for IP addresses and DNS entries. I ordered the larger server at a reverse auction, which is a bit unusual. The provider offers used and abandoned servers for monthly rent. Step by step, the price goes down until someone takes the offer. That made it relatively cheap.

One could do this with “dedicated servers” from other providers, but as soon as you have hardware such as 256 gigabytes of RAM, and multiple terabytes of hard disk, it gets expensive.

DANTE supports the project by paying the monthly server rent and fees. The costs for the domains are not included.

And now to the content, roughly sorted by the service type.

2 Internet forums

`LaTeX.org` is a classic LATEX Internet forum, in the English language. The TeXnicCenter developer Sven Wiegand founded it as `LaTeX-Community.org` in 2007. I joined as a moderator in 2008, took over the administration in 2012, and moved it to its own server. In 2017, I was able to register the domain `LaTeX.org`, and that is now what the forum is called. To date, 17,000 users have made 99,000 posts on 25,000 topics. The classic thread discussion culture lives on here.

`goLaTeX.de` was founded by Johannes Ähling in 2008 as a German-language forum for TEX. In 2015 I took over as admin for the purpose of updating the software and relaunching the forum. To date, 8,700 users have written more than 108,000 posts on 21,000 topics.

`TeXwelt.de` was started by me in 2013 as a German-language counterpart to StackExchange, for trying out this Q&A format and offering it to German-speaking users. So far, 1,300 users have written 3,700 questions and 4,200 answers. Here I would like to switch to the Askbot software or another alternative; maybe someone can help?

`TeXnique.fr` was created in 2016 as a joint project by Patrick Bideault, Denis Bitouzé, and me, with the support of the French TEX users group GUTenberg, as a question-and-answer site for LATEX in the French language. So far, 400 users have written 1,200 questions and 1,600 answers.

Each of those forums has a built-in online compiler, so you can translate code samples directly with a tap on your smartphone and view it as PDF output. You can also edit the code in the posts online and test the changes directly.

3 LATEX tools

`TeXlive.net` is the online compiler mentioned above. It was developed by David Carlisle and initially used at `LearnLaTeX.org` on a small free AWS instance. When that was no longer possible, we moved it to the DANTE-sponsored servers. The basis is an up-to-date and complete TEX Live system with all packages and tools such as BIBTEX, Biber, and `makeindex`. About 2,000 users translate around 3,000 to 4,000 documents every day, so I estimate over one million translated documents a year. This is also because `TeXlive.net` is integrated into the forums mentioned above, as well as into the blogs, wikis, galleries, and FAQ pages mentioned later.

`TeXdoc.org` is the online version of `texdoc`. It was started in 2014 as `TeXdoc.net` with a frontend written by Paulo Cereda, so that Overleaf users could easily access TEX documentation online without local documentation, or TEX users can read on the go on their smartphones when browsing forums. It comes with search features and with standardized links, and an API for external calls, such as from the forums. It now runs in a Docker container, which is being

Stefan Kottwitz

doi.org/10.47397/tb/43-3/tb135kottwitz-servers

further developed by the *Island of TeX* project that you can visit at `IslandOfTeX.org`.

`CTAN.net` was set up by me as a CTAN mirror in 2018 because the servers were there and had capacity. So it was straightforward to mirror CTAN in order to further distribute content and accelerate the installation load. And as Michael Doob said: "Having a … CTAN mirror will change your life."

`LaTeX2e.org` is a mirror of the "LaTeX 2ε Unofficial Reference Manual" from `latexref.xyz`, since the widest possible mirroring is desired.

4 Wikis and FAQs

`TeXfragen.de` was started by Patrick Gundlach as a new TeX FAQ as a wiki; several supporters contributed articles. In 2017, I took it over as admin and added selected excellent answers from `TeXwelt.de` to the wiki, among other things, and of course, added online compiling as on the other pages.

`goLaTeX.de/wiki` is another German-language wiki that documents LaTeX commands and packages and provides examples that can be tested online.

`LaTeX.net.br` is a Brazilian Portuguese version of the classic TeX FAQ. I migrated it with a translator in 2018 on the occasion of the TUG conference in Rio de Janeiro. Due to a lack of support, I did not develop it further afterward, but it is still online.

`TeXfaq.org` was hosted here, too, until recently, but it has now moved to Github. This was the original `www.tex.ac.uk` FAQ from Robin Fairbairns and many contributors. At some point, it could no longer be operated in Cambridge, so I took over hosting with Joseph Wright and David Carlisle in 2015. Later we changed the domain. `www.tex.ac.uk` no longer exists as a domain due to involved costs.

5 Blogs

LaTeX developers, authors, and users use blogs to occasionally publish articles about their own projects, experiences, or news from the TeX world. The following blogs are running on DANTE-sponsored servers.

`LaTeX.net` began as a collection of articles in a know-how area on `LaTeX-Community.org`, provides articles from many years, and now serves primarily as a blog for current news.

`TeX-Talk.net` is the TeX StackExchange community blog. It originated at `tex.blogoverflow.com` and was retired by StackExchange in 2016. As a moderator on StackExchange, I urged preserving it, and so the content was handed over to me to continue operating under a new name. The section with 20 in-depth community-moderated interviews of well-known TeX.SE users is particularly interesting.

`TeX.my` is the blog of a Malaysian LaTeX user group, started in 2009 and moved here in 2016. Today it is mainly written by Lian Tze Lim, whom we know as a primary Overleaf LaTeX advisor.

`cnltx.de` is Clemens Niederberger's blog. Because of its focus on chemistry, it previously ran under the name `mychemistry.eu`.

`TeX.co` is a TeX blog started by members of `TeXwelt.de`, which was later separated.

`TikZ.de` is a blog by some Ti*k*Z friends, including me. It's rather quiet in there at the moment because no one has much time. It's not abandoned and I have the best excuse, now writing a Ti*k*Z book.

`TeXblog.net` is my TeX blog from 2008, but I've put it so far behind everything else that I hardly use it at the moment.

6 Galleries

`TeXample.net` is a collection of Ti*k*Z examples and was built in 2008 by Kjell Magne Fauske. When Kjell no longer had time for it, I took over as admin in 2012. Today there are 407 submitted Ti*k*Z examples from 190 authors. There is also a blog aggregator at `TeXample.net/community`, a kind of news ticker from the TeX blog world. There are 3,600 entries from 50 registered blogs (including TUG's).

`TikZ.net` is another Ti*k*Z gallery that I designed as a potential successor of `TeXample.net` because the latter became difficult to maintain. Most of the approximately 1,000 examples have been written by Izaak Neutelings, with a focus on physics.

`TikZ.fr` is a French-language Ti*k*Z gallery that I'm currently building with Patrick Bideault, Denis Bitouzé, and Alain Matthes, with a focus on "tkz" packages such as `tkz-euclide`.

`pgfplots.net` is a gallery of selected plots that I set up as an enthusiastic `pgfplots` user.

`LaTeX-Cookbook.net` is a gallery of examples from my "LaTeX Cookbook" from 2015. All code examples of the book are available there, editable and compilable online.

`LaTeXguide.org` is the companion page to the "LaTeX Beginner's Guide", started on the occasion of the new edition in 2021 with updated code for all examples and additional information.

7 Thanks

I would like to thank DANTE e.V. for funding the hardware that makes possible what is described here. It is a great motivation for working on these projects.

◇ Stefan Kottwitz
Hamburg, Germany
`stefan (at) latex.org`
`https://latex.org`

The residual concepts of production vs. the emergent cultures of distribution in publishing

David Blakesley

Abstract

Who wins? The base or the superstructure? I'm not a Marxist per se, but I've lived this struggle for some time as a writer and publisher. In this essay, adapted from my TUG'22 keynote presentation, I describe my efforts to change or adapt the democratized tools of production to produce new forms of writing, which ultimately led to an ongoing battle with the dominant cultures of production in the world of publishing.

I'll narrate two case studies. One focuses on the writing and production of an innovative, if not disruptive, textbook in the ultra-conservative textbook industry. The second tells the ongoing story of an interloping publishing company (Parlor Press) that reveals the central challenge of *distribution* for both writers and publishers, from typesetting (print) to transformation (digital).

LaTeX developers and users, take note! The return of the nonbreaking space and soft return is nigh!

1 Some background

I'm the Campbell Chair in Technical Communication and Professor of English at Clemson. I also direct the PhD program in Rhetorics, Communication, and Information Design. That's my day job. My night job is as the founder and publisher of Parlor Press, a scholarly publishing company I launched in 2002. Originally, Parlor Press was created to publish just one book, a collection of letters exchanged between Kenneth Burke and William Rueckert, two people I've written about in my own scholarship.

One thing led to another, and now we've published 350 books in a variety of formats, from print to multimedia ebooks, and have eighteen book series, most in the humanities. I'm the only employee. I do hire some freelance editors now and then to help. Series editors help with development and acquisition, but I'm largely responsible for production, design, distribution, and marketing. And then there's shipping, metadata, customer relations, author support, a Shopify website, accounting, and more. I know. It sounds impossible, but I have learned over time to be efficient.

2 Opening questions and definitions

Who wins in the world of publishing? The base or the superstructure? I'm not a Marxist per se, but I've lived this struggle for some time as a writer and publisher. In this essay, which I've adapted from my presentation at TUG 2022 (available online via `tug.org/l/tug22-video`), I describe my efforts to change or adapt the democratized tools of production to produce new forms of writing, which ultimately led to an ongoing battle with the dominant cultures of production in the world of publishing.

I'll narrate two case studies. One focuses on the writing and production of an innovative, if not disruptive, textbook in the ultra-conservative textbook industry. The second tells the ongoing story of an interloping publishing company (Parlor Press) that reveals the central challenge of distribution for both writers and publishers, from typesetting (print) to transformation (digital).

I want to begin with some terms that I've used to try to explain what I do and understand how it has changed over time. The struggle for me has been with the culture of publishing. Here are some definitions of these key terms that help me understand my academic and publishing existence, with a few examples. I've always found Raymond Williams helpful, particularly his essay from *New Left Review*, "Base and Superstructure in Marxist Cultural Theory", from which I draw the definitions below. I provide the definition and then a visual example.

The base "'The base' is the real social existence of man. 'The base' is the real relations of production corresponding to a stage of the development of material productive forces. 'The base' is a mode of production at a particular stage of its development." (Williams)

Figure 1: The base in publishing. These are just a few of the means of production.

The superstructure "The superstructure consists of the cultural and economic forces that both reflect and maintain the material base, the mode of production. The superstructure is of a secondary order and symbolic." (Williams)

Hegemony "Hegemony is the expression of power, an ideological force that dominates social, cultural, and economic life and thus stabilizes the base, the modes of production." (Williams)

Example: The lingering hegemony of print. The superstructure supports print production and discourages challenges to the status quo.

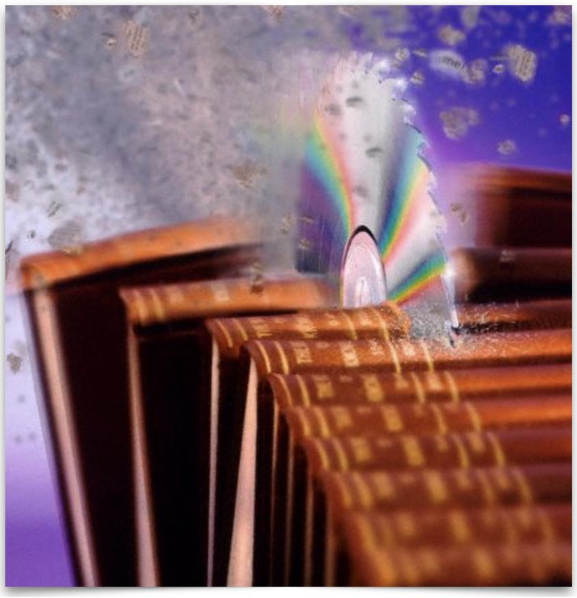


Figure 2: CD-ROM ebook destroys printed books. (Image by Chuck Savage. RF Corbis Collection. Getty Images.)

Dominant culture "The modes of incorporation are of great social significance, and incidentally in our kind of society have considerable economic significance. The educational institutions are usually the main agencies of the transmission of an effective dominant culture, and this is now a major economic as well as cultural activity; indeed it is both in the same moment." (Williams)

According to Williams, we can identify at least three cultural forces at work in any given social formation, such as publishing. The dominant culture of publishing is powerful, ubiquitous, largely impenetrable, persistent, and largely invisible.

Residual culture "The meanings and values which cannot be verified or cannot be expressed in the terms of the dominant culture, are nevertheless lived and practised on the basis of the residue-cultural as well as social-of some previous social formation." (Williams)

Residual culture is persistent, a hanger-on, and in some cases never disappears. The status quo has great momentum. Most academics feel its effects daily and especially if you try to change anything.

Emergent culture "New meanings and values, new practices, new significances and experiences, are continually being created. But there is then a much earlier attempt to incorporate them, just because they are part — and yet not part — of effective contemporary practice." (Williams)

Emergent culture is where I prefer to live. It is the future. The problem, said William Gibson, is that the future is already here, but it's just not evenly distributed yet. Much of my work as a writer and a publisher is to distribute the future, which sounds rather idealistic, I suppose.

Here's a nice example of residual and emergent cultures in contest.

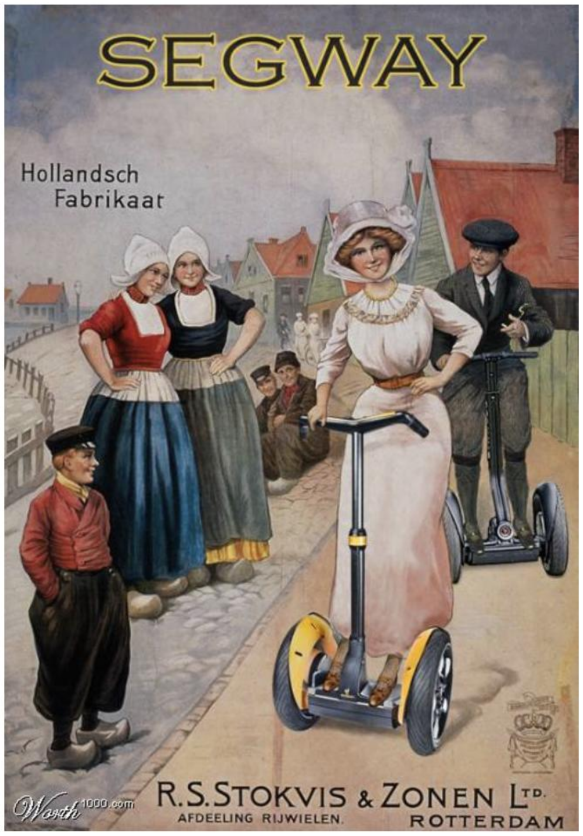


Figure 3: Residual and emergent cultures battle for power and attention. Was the Segway an invention or a cultural composition? (Image courtesy worth1000.com.)

3 Case study 1: Writing in the digital age

My first case focuses on my work as an author. I've written a textbook that has appeared in multiple editions, their covers shown in Figure 4.

These covers illustrate what changed over a period of about four years of work. The one on the left (*The Thomson Handbook*) was created by the publisher with absolutely no consultation with the

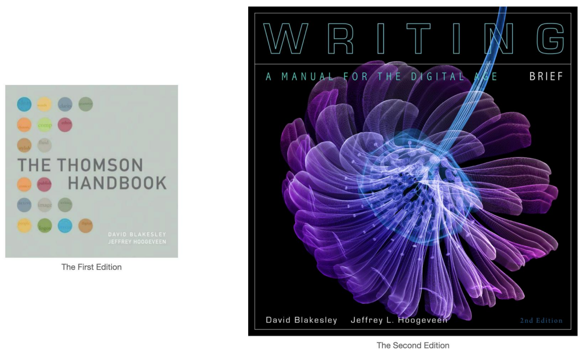


Figure 4: The covers of the first and second editions of the author's writing handbook, published by Cengage (2009 and 2012).

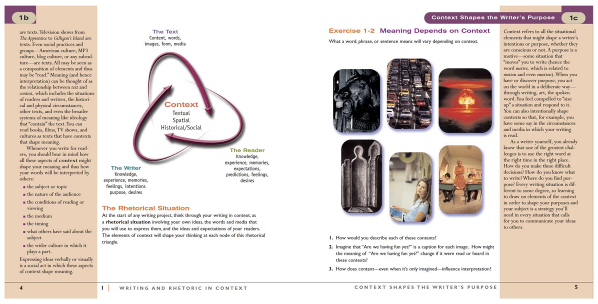

1b
Context
Textual
Spatial
Historical/Social
The Text
The Writer
The Reader
The Rhetorical Situation
Context Shapes the Writer's Purpose
1c
Exercise 1-2 Meaning Depends on Context
WRITING AND RHETORIC IN CONTEXT
CONTEXT SHAPES THE WRITER'S PURPOSE

Figure 5: The innovative design of *Writing: A Manual for the Digital Age* transforms the principles of the illuminated manuscript. Here, the (usually) primary text has been moved to the margins, with the visual content and illustrations occupying the center of attention on the inside.

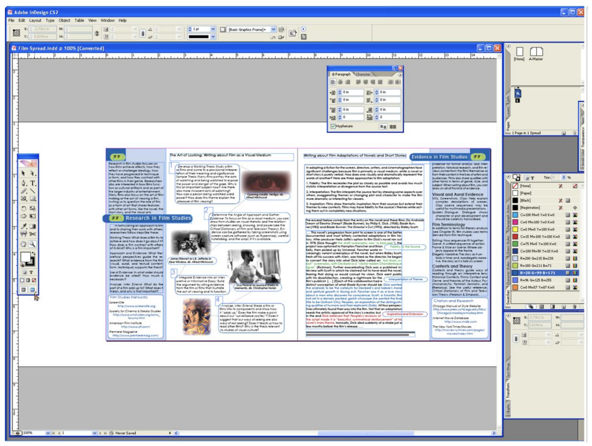


Figure 6: Composing in the design/production space as an author. The image shows the Adobe InDesign workspace.

authors and having little, if any, visual relationship with the content of the book. A bunch of dots with words in them. The cover on the right, is much different. I chose it and persuaded the publisher to go with it. Here's the caption on the inside front cover:

> Is a flower born digital? Macoto Murayama's are. Using his sketchpad, camera, microscope, and 3D rendering software, this artist joins technology with the craft of photography to help us see the natural world in a brand-new way.
>
> Can a writing handbook help students encounter the digital world in a brand-new way? Only if it joins the art of writing with the technology for enhancing it. Blakesley and Hoogeveen's *Writing* — finally, a rhetorical handbook for students born digital.

Of note here is that from the first to the second edition we see a shift from the publisher at the center of the production process in the direction of the author, who suddenly has some say in the means of production.

Figure 5 shows an interior spread. The visual content is primary, with the textual content functioning like an illumination. The full version of the first edition came in at 1,300 pages. Can you imagine writing, designing, and producing that?

It was critical for me to create (invent) in the design space, with the images and text in close proximity. The images and text converse with one another. MS Word doesn't do well with images and text together, so I composed much of the book in InDesign (CS2, here, about fifteen years ago). Figure 6 shows a screenshot of my workspace, designing what was called a research spread. Once composed, I produce a PDF to share with the publisher so that designers would see what I had in mind.

Then I had to retrofit the document so that I could share the content in a Word file because that's how the publisher's book designers were prepared to receive new content. Images were submitted as separate files. The 1,300 page book translated into a 7,000 page Word document, about a three-foot tall stack of paper (Figure 7).

Figure 8 shows what the finished version looked like in the first edition. Pretty nice! By the time we got to the second edition, I had persuaded the publisher (now Cengage) to produce all of the content in InCopy — a shared workspace version of InDesign. Cengage did not agree overnight. The dominant and residual cultures of textbook publishing kept authors a long way from the means of production. This new workflow changed that.

I'm sure this case wasn't a precipitating cause, but within a couple of years, the dominant culture of textbook publishing changed rather dramatically,

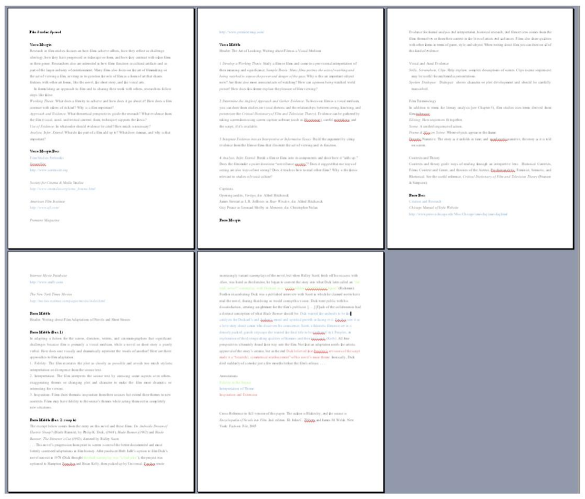


Figure 7: The InDesign document had to be disassembled into a Word document and sent separately with the images. A PDF file showed the layout.

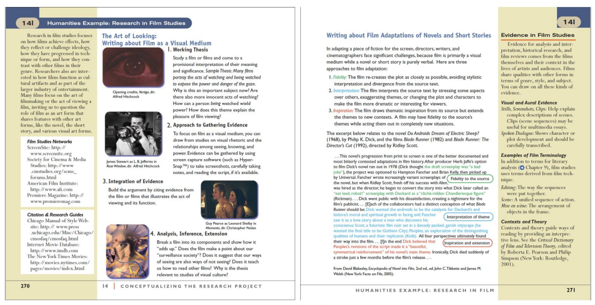


Figure 8: The interior spread as reassembled by the publisher's contracted production designer.

mostly due to economic forces (students stopped buying new textbooks, opting for rentals or nothing). Textbook companies are largely now content managers and distributors, with authors relegated to the margin, working on modules that can be redistributed across platforms. The textbook companies make their money with ancillaries. The base, where authors had a vital role in the means of production (invention), gives way to the superstructural forces of data management.

4 Case study 2: Publishing in the digital age

At different moments in the emergence of publishing, the power of different stages over the others rises and falls. For example, it used to be the case that marketing drove everything. In scholarly publishing around the turn of the twenty-first century, marketing and marketability started to compromise the whole process. Authors had to be well known, writing about popular and timely topics in books that would sell

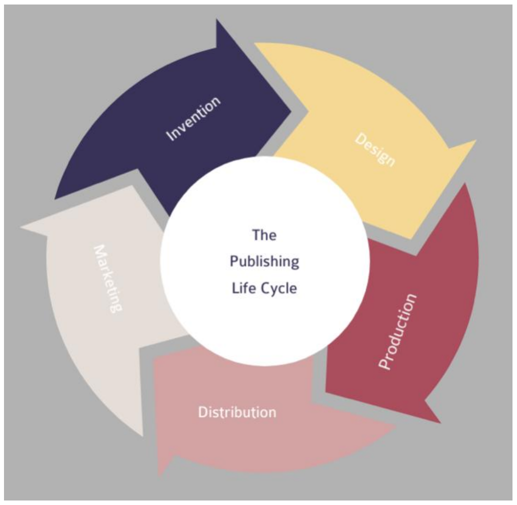


Figure 9: The publishing life cycle.

to more than libraries. Outside peer review took a back seat to evaluations by the marketing team.

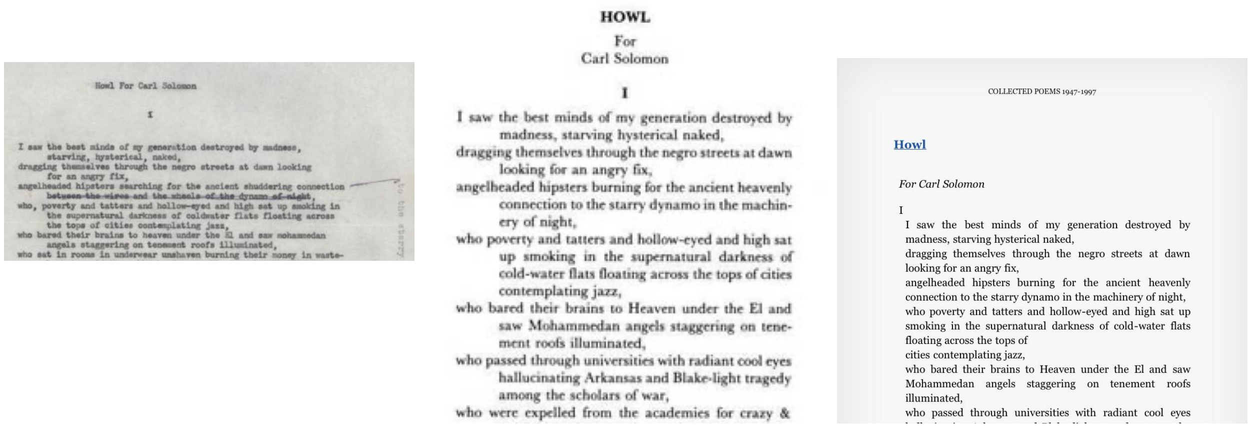
With the emergence of new formats, like EPUB and MOBI, the stress on production processes increased, which in turn meant that the desire for efficiency started to compromise production values. Amazon's Kindle promised to make books much cheaper to read, almost overnight, but the means of production couldn't keep up. Take the example of the first publication of Allen Ginsberg's *Howl*, which Amazon touted with much fanfare. However, the Kindle format (MOBI, a sort of simplified and proprietary version of EPUB) could not easily handle line spacing, line breaks, or stanza breaks (Figure 10). Some expertise in the conversion of print to EPUB or MOBI was suddenly required.

What people were saying

"I tweeted my frustration. Others did too. What does this say for eBooks if we can't get basic things like formatting right? Why create such hullabaloo around this digital release if you hadn't properly checked formatting on every device? Why is it that publishing sits so far outside the norms of what is required to launch something digital?"
—Callie Miller, *The Lit Life*, 7 Oct 2010,
`www.litlifela.com/counterbalance/2010/10/html-ebook-formatting-nonsense.html`

Some of the responses to Callie Miller's blog post are shown in Figure 11.

Parlor Press has published a lot of poetry, more than seventy titles so far (Figure 12). Originally, none of them was available in EPUB or Kindle format. I knew that we had to eventually convert all of our books to EPUB. So what were my options? I could



Howl For Carl Solomon

I

I saw the best minds of my generation destroyed by madness,
    starving, hysterical, naked,
dragging themselves through the negro streets at dawn looking
    for an angry fix,
angelheaded hipsters searching for the ancient shuddering connection
    ~~between the wires and the wheels of the dynamo of night~~,
who, poverty and tatters and hollow-eyed and high sat up smoking in
    the supernatural darkness of coldwater flats floating across
    the tops of cities contemplating jazz,
who bared their brains to heaven under the El and saw mohammedan
    angels staggering on tenement roofs illuminated,
who sat in rooms in underwear unshaven burning their money in waste-

to the starry

HOWL

For
Carl Solomon

I

I saw the best minds of my generation destroyed by
    madness, starving hysterical naked,
dragging themselves through the negro streets at dawn
    looking for an angry fix,
angelheaded hipsters burning for the ancient heavenly
    connection to the starry dynamo in the machin-
    ery of night,
who poverty and tatters and hollow-eyed and high sat
    up smoking in the supernatural darkness of
    cold-water flats floating across the tops of cities
    contemplating jazz,
who bared their brains to Heaven under the El and
    saw Mohammedan angels staggering on tene-
    ment roofs illuminated,
who passed through universities with radiant cool eyes
    hallucinating Arkansas and Blake-light tragedy
    among the scholars of war,
who were expelled from the academies for crazy &

COLLECTED POEMS 1947-1997

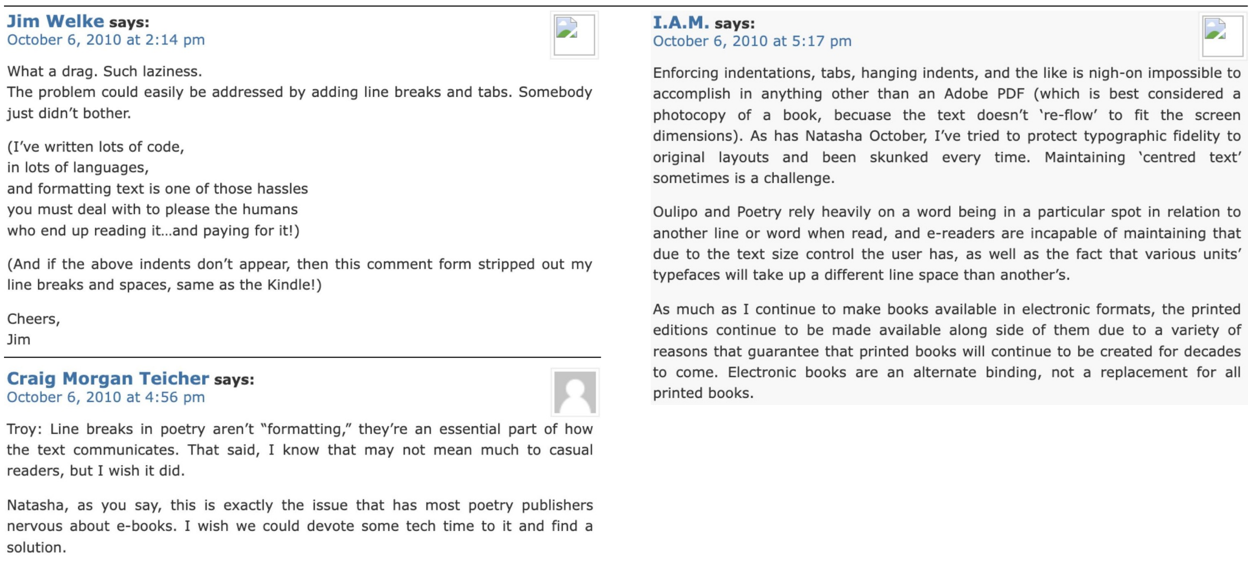
Howl

For Carl Solomon

I

I saw the best minds of my generation destroyed by madness, starving hysterical naked,
dragging themselves through the negro streets at dawn looking for an angry fix,
angelheaded hipsters burning for the ancient heavenly connection to the starry dynamo in the machinery of night,
who poverty and tatters and hollow-eyed and high sat up smoking in the supernatural darkness of cold-water flats floating across the tops of
cities contemplating jazz,
who bared their brains to Heaven under the El and saw Mohammedan angels staggering on tenement roofs illuminated,
who passed through universities with radiant cool eyes

Figure 10: The original typed manuscript of *Howl* (left), the print edition (middle), and the MOBI (Kindle) edition (right).



Jim Welke says:
October 6, 2010 at 2:14 pm

What a drag. Such laziness.
The problem could easily be addressed by adding line breaks and tabs. Somebody just didn't bother.

(I've written lots of code,
in lots of languages,
and formatting text is one of those hassles
you must deal with to please the humans
who end up reading it…and paying for it!)

(And if the above indents don't appear, then this comment form stripped out my line breaks and spaces, same as the Kindle!)

Cheers,
Jim

Craig Morgan Teicher says:
October 6, 2010 at 4:56 pm

Troy: Line breaks in poetry aren't "formatting," they're an essential part of how the text communicates. That said, I know that may not mean much to casual readers, but I wish it did.

Natasha, as you say, this is exactly the issue that has most poetry publishers nervous about e-books. I wish we could devote some tech time to it and find a solution.

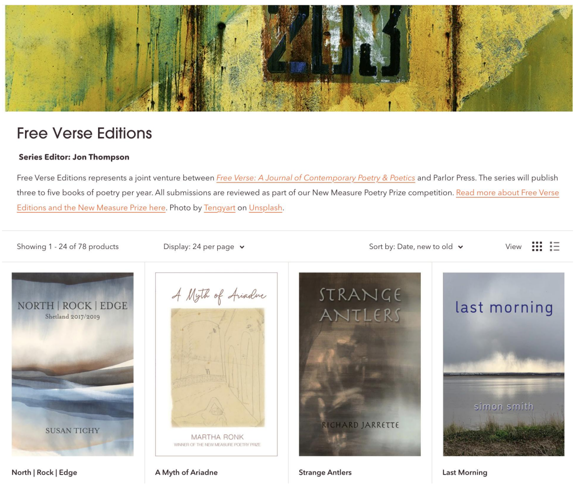
I.A.M. says:
October 6, 2010 at 5:17 pm

Enforcing indentations, tabs, hanging indents, and the like is nigh-on impossible to accomplish in anything other than an Adobe PDF (which is best considered a photocopy of a book, becuase the text doesn't 're-flow' to fit the screen dimensions). As has Natasha October, I've tried to protect typographic fidelity to original layouts and been skunked every time. Maintaining 'centred text' sometimes is a challenge.

Oulipo and Poetry rely heavily on a word being in a particular spot in relation to another line or word when read, and e-readers are incapable of maintaining that due to the text size control the user has, as well as the fact that various units' typefaces will take up a different line space than another's.

As much as I continue to make books available in electronic formats, the printed editions continue to be made available along side of them due to a variety of reasons that guarantee that printed books will continue to be created for decades to come. Electronic books are an alternate binding, not a replacement for all printed books.

Figure 11: Response to the original Kindle version of *Howl*.



Free Verse Editions

Series Editor: Jon Thompson

Free Verse Editions represents a joint venture between *Free Verse: A Journal of Contemporary Poetry & Poetics* and Parlor Press. The series will publish three to five books of poetry per year. All submissions are reviewed as part of our New Measure Poetry Prize competition. Read more about Free Verse Editions and the New Measure Prize here. Photo by Tengyart on Unsplash.

Showing 1 - 24 of 78 products   Display: 24 per page   Sort by: Date, new to old   View

North | Rock | Edge   A Myth of Ariadne   Strange Antlers   Last Morning

Figure 12: Some sample Parlor Press poetry titles. See `parlorpress.com/collections`.

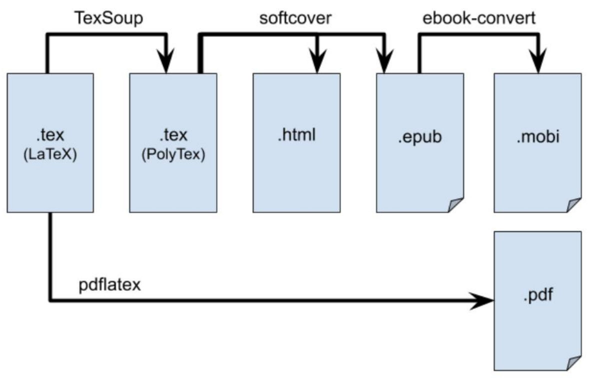


Figure 13: Ivan Savov's diagram of the LaTeX conversion process to multiple platforms. From Minireference blog: Starting a Revolution in the Textbook Industry, `minireference.com/blog/generating-epub-from-latex`.

pay Ingram, for example, to convert our backlist (now 350 titles), but at $200 or more a pop, we're talking about $70,000 or more to cover the conversion cost. Yikes. I tend to be a DIY person, know a little bit about coding, and have worked with open source projects like Drupal since its beginnings.

I had heard of LaTeX, but it looked quite daunting (Figure 13). Michael Hartl described the process in excruciating detail in the manual for his Softcover system (`manual.softcover.io`), when the source material includes math: The real challenge is producing EPUB and MOBI output. The process was to:

(1) create a self-contained HTML page with embedded math,

(2) include the amazing MathJax JavaScript library, configured to render math as SVG images,

(3) then hit the page with the headless PhantomJS browser to force MathJax to render the math (including any equation numbers) as SVGs,

(4) extract self-contained SVGs from the rendered pages, and

(5) use Inkscape to convert the SVGs to PNGs for inclusion in EPUB and MOBI books.

Easy, right? In fact, no — it was excruciating and required excessive amounts of profanity to achieve. But it's done, so ha. Piece of cake! I had to think about the conversion process in terms that were more familiar (Figure 14).

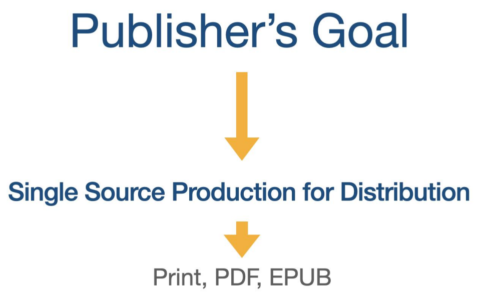


Figure 14: The challenge from the publisher's perspective.

I have to work with authors, and there's no reason why they shouldn't be aware that what and how they write should not to some extent reflect the means of production. Few writers worry about that, however, and do what the word processor encourages them to do (namely, format and design with abandon). Publishers force the issue, however, and ask authors to prepare manuscripts in very particular ways, most of which authors ignore. Production changes/affects invention/authoring. This was a lesson I learned from my textbook. Now, however, I realized that distribution should affect/change authorship as well. (Figures 15 and 16.)

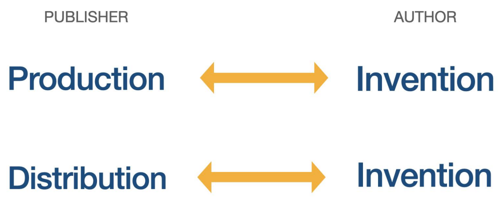


Figure 15: Publishers work with authors to help them prepare submissions for production.

The hegemony of spaces, tabs, and hard returns preserves the status quo of production as governed by residual and dominant cultures and embedded not just in software but in the socialized practices of the people. The nonbreaking space and soft return are elements of the emergent culture. Word processors and even keyboards encourage people to use spaces, tabs, and returns for line and paragraph spacing, regardless of the target format.

Authors, especially poets, need to understand how and why distribution changes their art. We have heard a lot lately about how much the supply chain is vulnerable to disruptive forces. So it has been with the emergence of new distribution lines and the new formats they deliver, like EPUB and MOBI.

Figure 17 shows the directions I now give our poets. They don't like it, but poets care about how their words are displayed. It matters, and makes a difference, even if subtle, for what their poetry means.

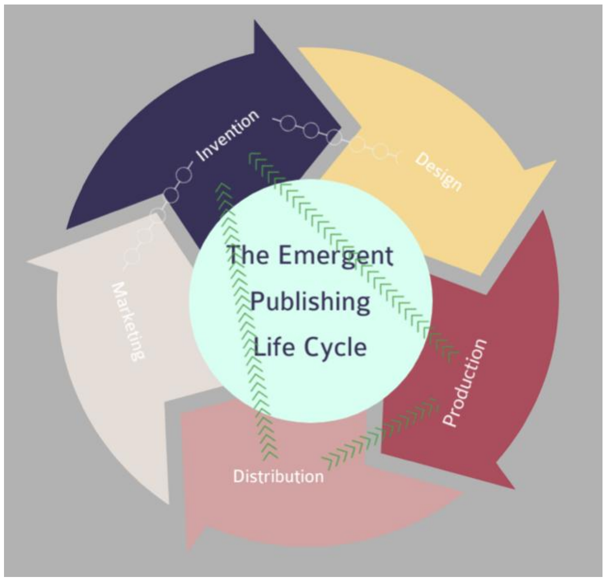
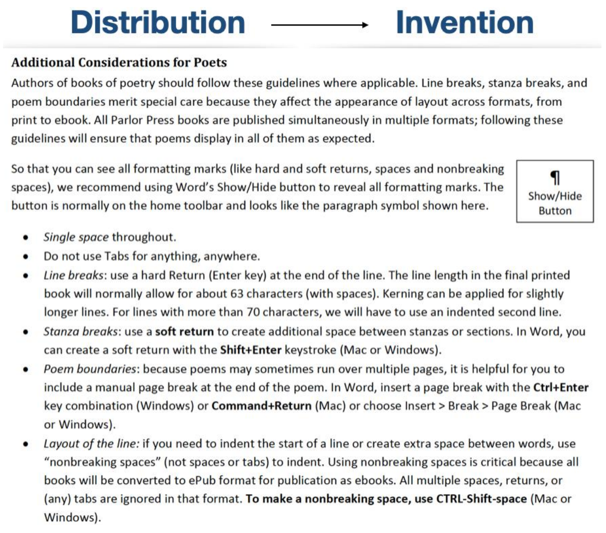


Figure 16: The lines of influence start to reach across the stages of the means of production.



Additional Considerations for Poets

Authors of books of poetry should follow these guidelines where applicable. Line breaks, stanza breaks, and poem boundaries merit special care because they affect the appearance of layout across formats, from print to ebook. All Parlor Press books are published simultaneously in multiple formats; following these guidelines will ensure that poems display in all of them as expected.

So that you can see all formatting marks (like hard and soft returns, spaces and nonbreaking spaces), we recommend using Word's Show/Hide button to reveal all formatting marks. The button is normally on the home toolbar and looks like the paragraph symbol shown here.

- *Single space* throughout.
- Do not use Tabs for anything, anywhere.
- *Line breaks*: use a hard Return (Enter key) at the end of the line. The line length in the final printed book will normally allow for about 63 characters (with spaces). Kerning can be applied for slightly longer lines. For lines with more than 70 characters, we will have to use an indented second line.
- *Stanza breaks*: use a **soft return** to create additional space between stanzas or sections. In Word, you can create a soft return with the **Shift+Enter** keystroke (Mac or Windows).
- *Poem boundaries*: because poems may sometimes run over multiple pages, it is helpful for you to include a manual page break at the end of the poem. In Word, insert a page break with the **Ctrl+Enter** key combination (Windows) or **Command+Return** (Mac) or choose Insert > Break > Page Break (Mac or Windows).
- *Layout of the line:* if you need to indent the start of a line or create extra space between words, use "nonbreaking spaces" (not spaces or tabs) to indent. Using nonbreaking spaces is critical because all books will be converted to ePub format for publication as ebooks. All multiple spaces, returns, or (any) tabs are ignored in that format. **To make a nonbreaking space, use CTRL-Shift-space** (Mac or Windows).

Figure 17: The directions for poets in the Parlor Press Author's Guide 2022.

So they are willing, much more so than authors of monographs.

The invention process of the author and the production process of the publisher has changed (Figure 18).

The screenshots in Figure 19 show three images of production. Authors originally used spaces, tabs, and multiple hard returns to manage their layout, which could be replicated in InDesign easily. However, to produce a properly formatted EPUB, you must use nonbreaking spaces and soft returns to manage line layout and stanza breaks. Figure 20

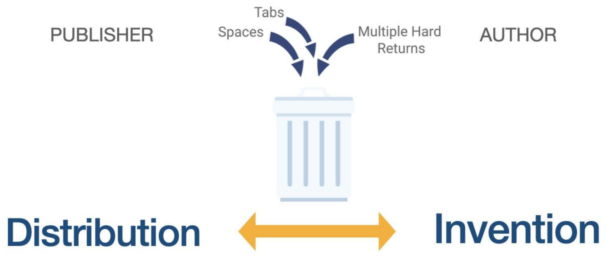


Figure 18: Distribution changes invention and production.

shows how the single-source now manifests in print and EPUB.

The problem was that all of our backlist titles had not been composed with the final platform in mind. Converting seventy poetry titles at four hours per book meant 280 hours of work (seven weeks!). Now, however, we produce an EPUB in ten minutes using the version prepared for print/EPUB simultaneously as a single source.

The work required to convert a backlist designed for print is substantial, particularly for a publisher like Parlor Press, which runs on a shoestring budget with no full-time employees. The democratization of production and digital printing technologies that made desktop publishing possible have led to new challenges at new stages of the publishing cycle. For the near future of publishing, the residual and dominant cultures of production must be reimagined in light of the emergent culture of distribution. Once that happens, the process of single-source development for multiple formats will be free and easy :).

5 References

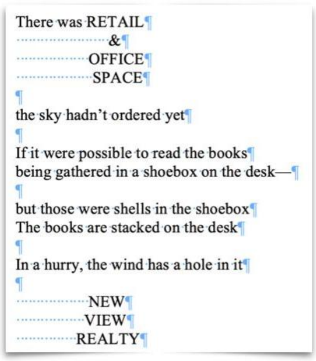
Hartl, Michael. The Softcover Book: Frictionless Self-Publishing. *The Softcover Book*, 2022. `manual.softcover.io`

Miller, Callie. Allen Ginsberg's Howl & eBook Formatting Nonsense (or, HTML is Hard). *The Lit Life*, 7 Oct 2010. `www.litlifela.com/counterbalance/2010/10/html-ebook-formatting-nonsense.html`

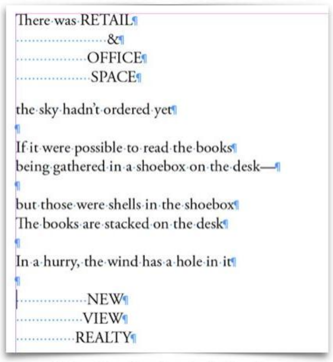
Savov, Ivan. Generating ePub from LaTeX. *Minireference blog: Starting a Revolution in the Textbook Industry*, 5 March 2021. `minireference.com/blog/generating-epub-from-latex/`

Williams, Raymond. Base and Superstructure in Marxist Cultural Theory. *New Left Review*, vol. 82 (Nov/Dec 1973), 319–348.

◇ David Blakesley
Clemson University and Parlor Press
`dblakes (at) clemson dot edu`

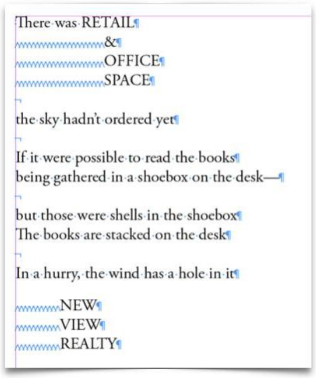

There was RETAIL¶
&¶
OFFICE¶
SPACE¶
¶
the sky hadn't ordered yet¶
¶
If it were possible to read the books¶
being gathered in a shoebox on the desk—¶
¶
but those were shells in the shoebox¶
The books are stacked on the desk¶
¶
In a hurry, the wind has a hole in it¶
¶
NEW¶
VIEW¶
REALTY¶

Author Version in Word


There was RETAIL¶
&¶
OFFICE¶
SPACE¶

the sky hadn't ordered yet¶
¶
If it were possible to read the books¶
being gathered in a shoebox on the desk—¶
¶
but those were shells in the shoebox¶
The books are stacked on the desk¶
¶
In a hurry, the wind has a hole in it¶
¶
NEW¶
VIEW¶
REALTY¶

Print Production in InDesign


There was RETAIL¶
&¶
OFFICE¶
SPACE¶

the sky hadn't ordered yet¶

If it were possible to read the books¶
being gathered in a shoebox on the desk—¶

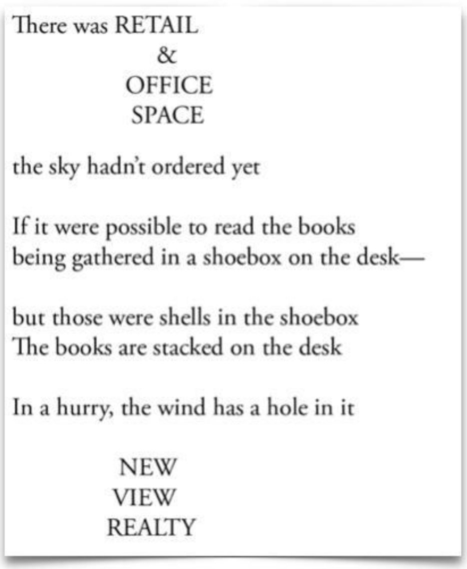
but those were shells in the shoebox¶
The books are stacked on the desk¶

In a hurry, the wind has a hole in it¶

NEW¶
VIEW¶
REALTY¶

EPUB Production in InDesign

Figure 19: Some examples, moving from the author through the production process, with EPUB now a target as well as print and PDF.


There was RETAIL
&
OFFICE
SPACE

the sky hadn't ordered yet

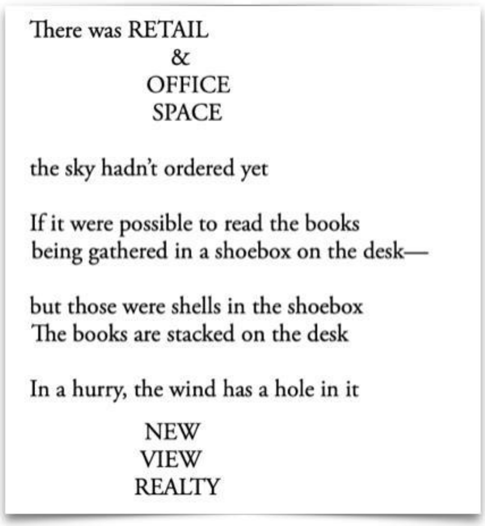
If it were possible to read the books
being gathered in a shoebox on the desk—

but those were shells in the shoebox
The books are stacked on the desk

In a hurry, the wind has a hole in it

NEW
VIEW
REALTY

PDF / Print


There was RETAIL
&
OFFICE
SPACE

the sky hadn't ordered yet

If it were possible to read the books
being gathered in a shoebox on the desk—

but those were shells in the shoebox
The books are stacked on the desk

In a hurry, the wind has a hole in it

NEW
VIEW
REALTY

EPUB in Ebook Reader

Figure 20: Output in print and EPUB are virtually identical once the tools of composition change.

Typographers' Inn

Peter Flynn

Font packages: Symbats, Swashes

Like many LaTeX users, I often check the LaTeX Font Catalogue [5] for a suitable typeface before deciding to buy or download one. Getting one of the listed fonts means there is often a package to support it, which makes life easier, especially when there are many variants.

Just as often, though, I need a font that isn't in the Catalogue, so I usually buy it — I'm happy to download fonts that the designer has explicitly made available free of charge and free of restrictions, but I'm not a great fan of downloading from sites that steal the work of designers and pass the fonts off as 'free'. Call me old-fashioned but font design is *hard*, and font designers need to pay the rent like everyone else.

Symbats. However, fonts made freely available by their designer are an act of generosity, and I came across one that was recently updated and released under the SIL Open Font License. I hadn't seen the font before, despite its having been available for 25 years! It's a font of religious and astrological symbols, not a text font, although it does also include Runic and Ogham alphabets (see Figure 1).

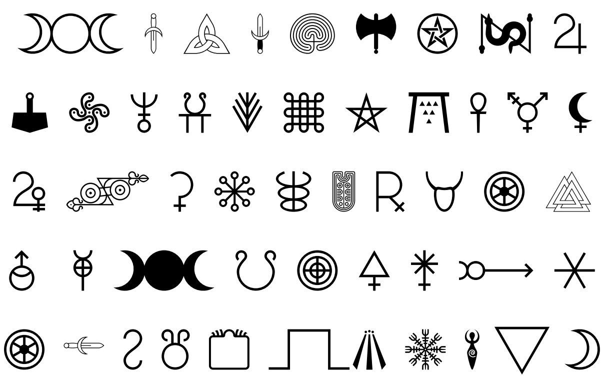


Figure 1: *Symbats 3.0* font (sample)

Those who remember the earlier days of fonts will recall that symbol font designers tended to put the symbols in the positions of the keyboard characters, so that A–Z, a–z, 0–9, and punctuation could easily be used to get the characters printed. But with so many symbols now having a standard Unicode name and codepoint, and with the big improvements in font file technology, this positioning is no longer needed in synchronous typographical editors. It's not needed for LaTeX editors either; for example ♒ (Aquarius) or ⚳ (Ceres) are both normal Unicode characters and can be typed as such if your editor has font support for them. However, some do not, so it seemed like a good step to write a package that would provide LaTeX commands for all the symbols in this font.

The first thing to do was to extract all the font character metadata into a spreadsheet using a combination of the *otfinfo* utility[1] and the *FontForge* scripting language. This provided one line per glyph, giving the actual character, the hexadecimal Unicode codepoint, and the Unicode or other name from the font file. I added a fourth column to provide a LaTeX command name. From this, it was possible to output a set of declarations like the one in Figure 2 which equated a command internal to the package (beginning with `SYMBATS@`[2]) with the `TU` font encoding and the codepoint of the symbol. As this was an OTF font, I was using XeLaTeX and the fontspec package, for which `TU` is the appropriate encoding (using XeLaTeX or LuaLaTeX is therefore a requirement for using the symbats3).

```
\DeclareTextSymbol{\SYMBATS@ceres}{TU}{"26B3}
```

Figure 2: Example of implementing an OTF glyph as a LaTeX internal command

I could have just used the LaTeX commands I had created in the spreadsheet, but some experiments had already shown that the positioning of the glyph with respect to the baseline of surrounding text did not take account of the depth (descenders) of the surrounding font. To allow for this, an additional definition would be needed for each character to expose the user command with code for the appropriate repositioning. I wrote a utility command `\SYMBATS@getdesc` to get the depth of the descenders of the current font and lower the baseline of the character by that amount (see Figure 3).

```
\newcommand{\ceres}[1][\relax]{%
  \SYMBATS@getdesc{#1}%
  \raisebox{\SYMBATS@baselineadjust}
           {\symbats\SYMBATS@ceres}%
}
```

Figure 3: Example of implementing an internal command as a LaTeX user command with an option to adjust the baseline (code in package)

As not everyone may want this, it is implemented by a package option `[descenders]`. Without it, a user can still add an optional argument to any of the character commands to raise or lower the symbol by an arbitrary amount.

[1] Available from CTAN in the lcdf-typetools package.

[2] Thanks to Karl Berry for the change from `SY@`.

A final step was to put the `\newfontface` font load command into `\AtBeginDocument` and add the font load option `[Scale=MatchUppercase]` so that symbols would be approximately the same size as the uppercase letters of the main font.

There are 325 symbols in the symbats3 package, which is available from a CTAN server near you (`https://ctan.org/pkg/symbats3`); it's also included in TEX Live and MiKTEX.

Swashes. Encouraged by this, I decided to repeat the process for another font that had been recommended to me. This is Super Tramp,[3] from designer and photographer Vivian Dehning.[4] It's a quite slender display font with a large collection of swash and variant characters, including a lot of extra ligatures, some on capitals, and a number of different upper- and lower-case asterisks.

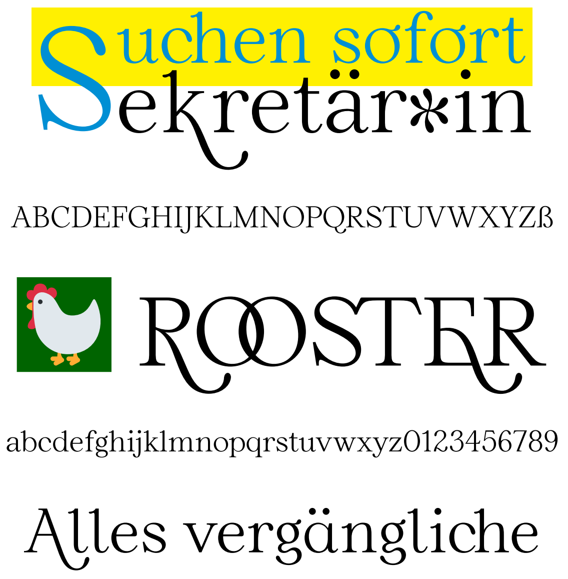

Suchen sofort
Sekretär*in
ABCDEFGHIJKLMNOPQRSTUVWXYZß
ROOSTER
abcdefghijklmnopqrstuvwxyz0123456789
Alles vergängliche

Figure 4: Vivian Dehning's *Super Tramp* font (sample, with chicken from the twemojis package).

Again, I thought it best to defer loading of the fonts until the last thing, so it could be adjusted to the main font size, although this time it uses `[Scale=MatchLowercase]`. Many of the additional characters use a StylisticSet option (up to eleven of them) in the OTF font, and there are ligatures and case-specific variants as well, so the font is currently loaded 13 times, which is probably wrong, but it works.

```
\newcommand{\Rswash}{{\ST@i R}}
\newcommand{\OOlig}{{\ST@xii OO}}
\newcommand{\Eswash}{{\ST@i E}}
```

Figure 5: Generated definitions for the *Super Tramp* font.

LATEX command names were incorporated into a spreadsheet as for the Symbats3 font, attempting to strike a balance between following the designer's choice of codepoint names and the conventions to which LATEX users have become accustomed. This resulted in some 200 commands, but this time they all seemed to sit on the baseline quite happily, so there was no need for adjustment, only a direct equivalence between LATEX command and the character from the correct StylisticSet or ligatured font load.

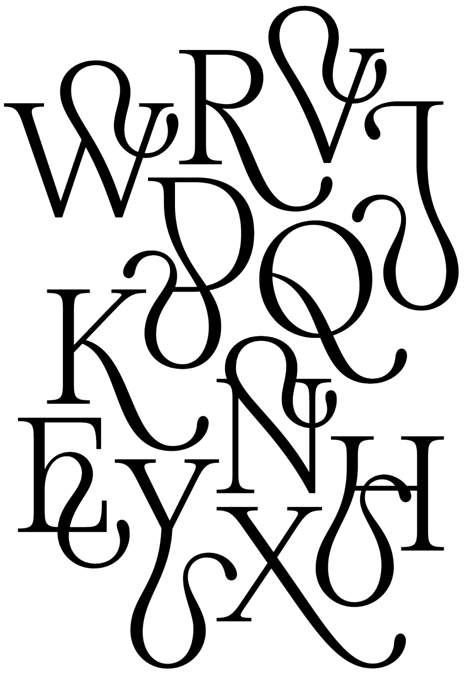


Figure 6: More sample swash characters from Vivian Dehning's *Super Tramp* font (designer's image, used with permission).

The naming of multiple swashes for the same base character led to the adoption of the same mechanism used for step-size fonts (`\large`, `\Large`, and `\LARGE`) so we have `\Jswash` (J), `\JSwash` (J), and `\JSWASH` (J), etc, in increasing order of complexity.

[3] Nothing to do with the rock band, movie, or trampoline manufacturer.

[4] This is a commercial font which can be bought from the designer via `https://viviandehning.com/super-tramp-typeface/`.

Characters that can be generated from the keyboard, especially the common diacritics, have *not* been given command names. For example, to get Ř you just type Ř using whatever mechanism your operating system and keyboard provides: there is no `\Rcaron` command; similarly for the Ẅ, but the crossed-W variant with diaeresis Ẅ is provided for with `\Wuml`.

There is still work to do in tidying this up before I can package it up for CTAN but it should be ready by the autumn.

To publish or not to publish

There have been some discussions recently among TeX consultants about providing a better service for self-publishers. These are authors — of all kinds of material, not just math and science — who for one reason or another are not using traditional publishers, so they have to edit, typeset, proof, bind or package, and publish their PDF or EPUB themselves.

There are of course lots of useful web sites, services, and other resources (including books!) to help people do this, many of them providing very good quality — at a cost — and very sound advice, often starting with 'Don't . . . '

The conventional path — a wordprocessor, often Microsoft Word — is perfectly capable, in expert hands, and with a lot of effort and reconfiguration, of producing passable typesetting. LaTeX can do the job just as easily if suitable classes and packages are used, and can produce better quality output.

I have always considered that LaTeX's key feature was automation: define a pattern once, and it can be used consistently throughout the whole document, and even across documents. Anyone who has ever taught LaTeX knows that avoiding repetitive manual formatting provides an instant productivity boost to the bottom line in terms of time saved. Newcomers have to rely on experienced users to create these patterns (the definitions of commands and environments), so for this project to succeed it is essential that we can open and maintain a dialogue between the potential users (authors with a book to typeset) and the potential contributors (experienced users).

The discussions include a TeX Live book publishing resource, and one of the tasks will be to promote it among self-publishers who care about publishing trade-quality books. One of the participants, the author Lloyd R Prentice [6], has said 'the big challenge is how to simply and clearly explain why LaTeX is a better book styling and typesetting option relative to word processing'. I think that's something everyone can contribute to.

Afterthought: Translations

I am always happy to be able to announce when an article from the Inn is translated into another language so that it reaches a wider audience. To be able to announce two translations and a reprint of the same article at the same time is even nicer.

My article 'To print or not to print' from this column [2] is now available in French as 'Imprimer ou ne pas imprimer' in the French typographical journal *Graphê* 85 [1], and was reprinted in *Cahier* 58 of GUTenberg (the journal of the French-speaking TeX users' group), translated by Patrick Bideault [4]; and in collaboration with Bernd Raichle in German as 'Druck oder Nichtdruck' in *Die TeXnische Komödie* (the journal of the German-speaking TeX users' group) [3].

I was remiss, however, in not RTFM and referring them to the copyright and translation notice inside the front cover of issues of *TUGboat* to ensure that *TUGboat* itself was notified of the publication. Thank you, Barbara Beeton, for reminding me.

References

[1] P. Flynn. Imprimer ou ne pas imprimer. *Graphê* 85:13, Dec 2020. Tr. Patrick Bideault. `https://typo-graphe.com/produit/numero-85/`

[2] P. Flynn. Typographers' Inn: To print or not to print. *TUGboat* 41(3), Dec 2020. `https://tug.org/TUGboat/tb41-3/tb129inn.pdf`

[3] P. Flynn. Druck oder Nichtdruck. *Die TeXnische Komödie* 2022(2), May 2022. Tr. Patrick Bideault and Bernd Raichle.

[4] P. Flynn. Imprimer ou ne pas imprimer. *Cahier* 58, Apr 2022. Tr. Patrick Bideault. `https://www.gutenberg-asso.fr/Cahier-numero-58-Septembre-2021`

[5] P. Jørgensen. The LaTeX Font Catalogue, May 2021. `https://tug.org/FontCatalogue/`

[6] L. Prentice, V. Novotný. *Publish Beautiful Books with Markdown.* Writersglen Publications, Marshfield, MA, 2021.

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

Formatting mesostic poems à la John Cage

David Bellows

Abstract

LaTeX is useful not only for producing beautifully typeset math and science papers but can be just as useful for the typesetting demands of modern poetry. This includes a style of poetry called *mesostics* that was created by the poet Jackson Mac Low and embraced and extended by the late composer John Cage; it is Cage's approach to mesostics that I will be addressing in this paper. I will show how I use LaTeX to automatically format my own computer-generated mesostics.

1 Introduction

Mesostics are a type of poetry and formatting that the composer John Cage used throughout the latter part of his career to create poetry by "reading through" a large source text looking for words containing the letters of the *spine* or *mesoword*, where both the source text and the spine were chosen for their significance to him. The resulting text would then be formatted in a way similar to an acrostic but with the spine word running through the middle. An example demonstrates this best:

```
          nearLy napping,
            camE a
         tappiNg,
            as Of
      gently Rapping,
  at my chambEr door.
```

In the above example (generated by my software), the source text is Edgar Allan Poe's *The Raven.* The spine is the name "Lenore" (from the poem) and can be seen as being the only letters in caps and running down the middle.

The software I use goes through whatever source text is given looking for words in the order they occur in the spine word (e.g., "Lenore") and selects extra words on either side of the spine word according to various rules and options. It then formats the results in one of three styles. The *regular* style, as you see above, was by far the most common style Cage used.

My software, The Platonic Music Engine (`www.platonicmusicengine.com`) (PME), exists to recreate all artifacts of human culture algorithmically with a high degree of human interaction, allowing the user to create unique works that bear a superficial similarity to existing ideas.

These artifacts of culture include music, visual art, poetry, divination, gaming, and anything else I can come up with.

In this case my software is being used to generate mesostics based on the user's choices (source text, spine, various styles of formatting, various other rules and options) and then printing out the result using LaTeX.

My software does all the heavy lifting. Not only does it find the mesostics, it also generates a `tex` file with all the necessary formatting in place. A better TeXnician would offload more work onto (La)TeX but I am not that person. I am also not much of a programmer; I just barely manage to get things working. Nonetheless, this particular workflow works very well for me. I generate the results the user wants and then use various external programs like LaTeX, Csound, LilyPond, etc., to handle all the output and make me look good.

All the programs that I use for output compile their results from text files. This is an extremely handy method of generating content.

The PME is released under the GNU Affero GPL v3 and can be downloaded from the above website. I welcome any additions to or comments about the software.

2 Regular style of mesostic

In my initial attempts at formatting, I used the `alltt` package (such as in the above example) and had my software figure out the spaces needed at the beginning of each line to make everything work out.

Since `alltt` uses a monospaced font, the process was pretty straightforward. Unfortunately it doesn't match Cage's published versions which used non-monospaced fonts and still managed to get everything perfectly spaced. I do not know how this was done in the various books Cage published that contained mesostics, but I'm guessing that a typesetter manually adjusted the letters as needed.

Since the goal of my software is to create content without the user having to tweak any of the output, I had to use an automated method but something better than what I was doing.

After struggling on my own for a while, I did what many of us do and asked for help from the TeX forum at stackexchange.com. Several people provided some great help that automated the formatting of mesostics but then I ran into various problems.

The overarching problem was that I couldn't understand those solutions at all. Anything beyond commands like `\textit` and there's a good chance I'm not going to be able to follow your solution.

The two basic problems I ran into is that unlike with Cage's original formatting, I wanted to allow

doi.org/10.47397/tb/43-3/tb135bellows-mesostic

the spine letters to be in bold. I feel that this helps them show up better.

I also noticed that in the *regular* style, Cage had the spine letters perfectly centered where the center of each letter ran down through the center of the other spine letters (you can typically only see this with the letter "I" but it does depend on the font being used). But in a different style, the left side of each spine letter was aligned with the other spine letters (see the *Merce Cunningham* style below).

I could not figure out how to make both of those changes in any of the offered solutions and I didn't want to keep bothering people about this stuff. So I went back to trying to figure it out myself.

I couldn't get tables to work. I can't remember why, but the kind of formatting I needed was too fine and I just couldn't get the level of control I needed.

So then I started using various multi-column packages. These worked better. The idea is that all the words to the left of the spine letter would be in one column set to flush right and then the spine letter and the rest of the words would be in the right column with no space between the columns.

I ran into the same problems as before with centering and bold faced fonts until I found a rather old package, `parallel`. In addition to separating the page into two columns, it makes it easy to use custom widths for the columns. See figure 1 for an example generated by my software.

near**L**y napping,
cam**E** a
tappi**N**g,
as **O**f
gently **R**apping,
at my chamb**E**r door.

Figure 1: Example of a PME mesostic using the regular style.

Here is the code for the first line:

```
\newlength{\charwidth}
\setlength{\charwidth}{\widthof{L}}
\begin{Parallel}{.5\textwidth - .5\charwidth}
     {.5\textwidth + .5\charwidth}
  \ParallelLText{\hspace*{-5cm}\raggedleft{near}}
  \ParallelRText{\textbf{L}\mbox{y napping, }}
\end{Parallel}
```

The software uses LaTeX to calculate the width of each spine letter ("L", in this example) and sets this value to the variable `\charwidth`. This value is then used to work out the space between the two columns such that the center of the letter is in the center of the page. Each line uses this same approach so that the spine letter is always perfectly centered regardless of its width.

I added some negative `\hspace` to the left column to allow any extra long lines before the spine letter to run off the page instead of breaking at the end of the line and messing up the formatting. Cage imposes a 45 character limit on either side of the spine letter (which my software follows) which should limit this from happening with this style, but see below for where it became an issue.

For some reason the `\hspace` hack wouldn't work on the right side of the column where I had to use `\mbox` instead. And I couldn't get `\mbox` to work on the left side, thus this hybrid solution.

3 Merce Cunningham style

John Cage created a set of mesostics for his partner, the dancer and choreographer, Merce Cunningham. For this one set of mesostics he devised a different style of formatting.

The basic idea was the same but each letter of each mesostic was subject to chance operations determining the typeface to be used for it, the font, size, weight and so on. According to Cage's description of the process, he had available to him over 700 typefaces with which to generate these mesostics.

Cage made each letter touch each other horizontally and spaced things in such a way that each line would touch the line before and the line after it. Cage intended for the resulting mesostics to look like Cunningham dancing, but felt like they ended up looking more like waterfalls. See figure 2 for an example published by Cage; the spine used in this mesostic is "Cunningham". Cage did not provide an unstylized version of the text and was probably fine with this particular style of mesostic being illegible as well as mostly incomprehensible.

I chose this example deliberately because it shows more overlap and collisions between the lines than most of these mesostics. I haven't studied the entire collection in depth but it appears that having the dot on the 'i' overlap is the most common type of collision.

See figure 3 for an example of how my software generates this style of mesostic using the same source text and spine as in my previous example. As in Cage's versions, there are no included words to the sides of the spine word.

The code for the first line:

```
\usepackage[letterspace=-150]{microtype}
...
\begin{Parallel}{.5\textwidth}{.505\textwidth}
 \linespread{2}\selectfont\lsstyle
 \ParallelLText{\hspace*{-10cm}\raggedleft
   {{\fontsize{11}{1em}\selectfont
```

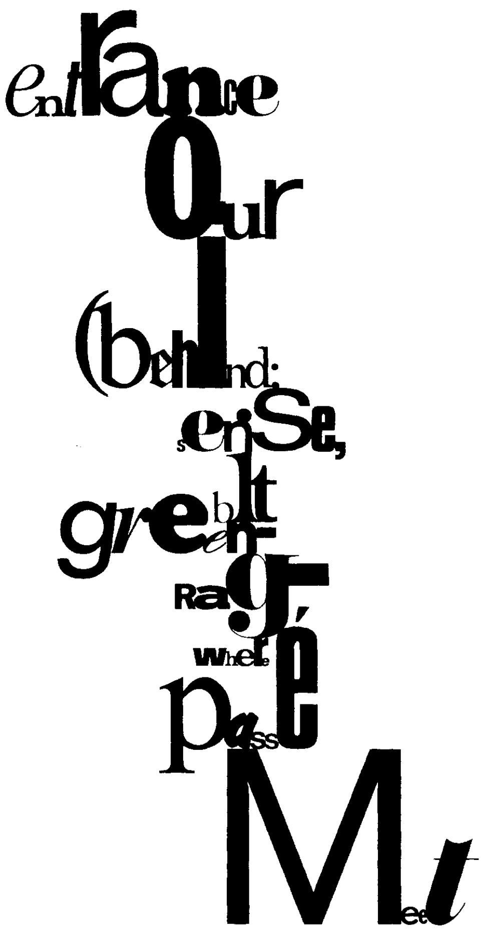


Figure 2: Example of a mesostic in the Merce Cunningham style, by John Cage. The spine is ‘Cunningham’. (Page 150 from *M: Writings ’67–’72* © 1973 by John Cage. Published by Wesleyan University Press. Used by permission.)

```
        {\textrm{\textit{n}}}}%
    {\fontsize{13}{1em}\selectfont
        {\texttt{\textit{\textbf{e}}}}}%
    {\fontsize{11}{1em}\selectfont
        {\textrm{\textit{a}}}}%
    {\fontsize{14}{1em}\selectfont {\texttt{r}}}%
   }}
 \ParallelRText{{\fontsize{36}{1em}\selectfont
   {\textsf{\textit{\textbf{l}}}}}%
 \mbox{{\fontsize{23}{1em}\selectfont {\texttt{y}}}%
 { }%
}}
\end{Parallel}
```

In the preamble for this style, you see my loading of the microtype package with a \letterspace of −150. I chose this number through experimentation and it seems to work well, getting the letters to touch each other horizontally but without overlapping too much. I expect that different typefaces might require different values.

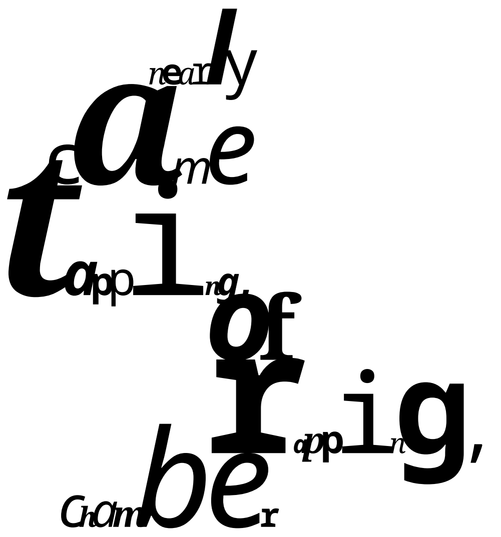


Figure 3: PME version of a mesostic in the Merce Cunningham style.

The \linespread command is calculated in my software based on the point size of the tallest letter in each line and multiplying that by a “magic” number that is user configurable. I experimented and settled on what I think is a good default value but the user is allowed to change this value if they want more or less space between each line which will decrease or increase the number of collisions and overlap between lines.

Another subtle change in these mesostics over the regular ones is that the spine letters, while still in the center of the page, are no longer perfectly centered with each other. Instead, the left sides of each letter are lined up and in the center of the page. If you look closely at figure 2 you can see this.

I didn’t come up with any cool mathematical way to make this work so I just have the left column end at the halfway part of the page. This should mean that the spine letter starts at that point. As you can see in figure 3, this doesn’t always line up perfectly. I don’t know how to fix this within my software or via (LA)TEX, so I might just have to be satisfied with close enough.

You can also see in the code that I extend the \hspace well into the left margin. Because of the potential size of the letters in this style, you will get examples that go off the page on both the right and left sides. On an aesthetic level I think this is okay and Cage would have been fine with it.

One of the interesting problems I ran into with this style was that using different typefaces caused all sorts of spacing issues. For example, combining Garamond with Helvetica created inconsistent spaces between letters and made centering the spine letter in this style much worse. So the unfortunate solution is to use only one typeface (Cage used many typefaces). For these examples I used Noto, which is freely available via a LaTeX package. It comes with both serif and sans serif fonts along with italics, bold, and bold italics. Other typefaces could be used but I just happen to like how this one looks by default.

The rest of the code works as with the previous example. On a personal note, I am very proud with how well this particular style turned out. It is not as elegant as Cage's versions, but given the limitations imposed on me by my skills and the nature of the project, I think my software produces pretty convincing results.

4 Ezra Pound style

The final style that I recreated from Cage is based on mesostics he generated by using Ezra Pound's *Cantos.* This was the easiest of the three. See figure 4 for an example of how my software generates mesostics in this style. I changed the spine to "Poe" to make the mesostic fit better in this publication.

se**P**arate dying ember wr**O**ught its th**E** floor
Presently my s**O**ul gr**E**w

Figure 4: PME version of a mesostic in the *Cantos* style.

The basic idea is that each mesostic now occupies a single line that is flush right. You still have the capital letters spelling out the spine, but they are no longer aligned vertically.

```
\begin{flushright}
 \normalsize
  se\textbf{P}arate dying ember wr\textbf{O}ught its
   th\textbf{E} floor \linebreak
  \textbf{P}resently my s\textbf{O}ul gr\textbf{E}w
   \linebreak
\end{flushright}
```

Probably the most interesting thing going on here is that the code uses the `\normalsize` command. In order to try to make sure each line fits on the page, my software adjusts the font size based on the number of characters being used in the line. This is calculated once based on the longest line and then is used for all the lines to ensure a consistent look.

5 Final thoughts

Formatting Cage's mesostics for my software was a challenge. It was a fun challenge and one where I believe the results justify the effort that went into it.

I wish I were better with (LA)TeX and could turn this into a package or figure out how to add it to an existing package, as I hope someone, someday might have a need for it.

I think there are two interesting ideas to take from this paper. One is that (LA)TeX, despite stereotypes to the contrary, can be extremely useful outside of STEM fields. It is an excellent tool for people working in the humanities and the arts.

The other idea is that much can be accomplished by people who don't have a tremendous amount of technical knowledge. I don't know what I'm doing most of the time but still manage to achieve my desired results. Obviously this is because of how well designed TeX and friends are and, of course, the amazing ecosystem that has grown up around it over the decades.

If I can do this stuff so can any other artist, composer, poet, gamer, ...

Finally, based on a suggestion from *TUGboat* editor Karl Berry, I am including one final mesostic (figure 5). This one uses the text of this article for the source text and the word "TUGBOAT" for the spine.

i ran in**T**o
always line **U**p perfectly.
formattin**G** of mesostics
at my cham**B**er
each letter **O**f
p**A**per).
ver**T**ically.

The spine
be in one col**U**mn set to
separate dyin**G**
on the num**B**er
s**O**urce text is
embr**A**ced
my desired resul**T**s.

Figure 5: Mesostic using this paper as the source text and the word "TUGBOAT" as the spine.

◇ David Bellows
davebellows (at) gmail dot com
https://www.platonicmusicengine.com

Representing Parkosz's alphabet in the Junicode font

Janusz S. Bień

Abstract

The 15th century Latin manuscript containing a treatise by Jakub Parkosz was the very first proposal of Polish spelling. To account for all the phonemes of Polish some new letters were proposed, which are not available in present day fonts. This makes it difficult to quote the proposal when discussing the history of Polish spelling. A transliteration was designed which uses only the characters available in the Unicode standard, but it was rather cumbersome. Another approach, suggested by the present author and implemented by Peter S. Baker in his Junicode (version two) font, is to use so-called tag characters.

1 Introduction

> A digression: when in 1990 I was writing my proposal of an extended font layout [3] and wanted to include some comments about the origin of the letters specific to the Polish language, I was very much surprised by the lack of published information on this topic. Now the situation is definitely better in one respect: the original sources have been digitized and are freely available to anybody. In my opinion the analysis of sources is still lacking many details, so some time ago I decided to try to answer my questions myself ☺.

The 15th century handwritten Latin treatise by Jakub Parkosz (called also Parkoszowic) was the very first proposal of Polish spelling. You can find the scans at (for example) `jsbien.github.io/Parkosz4IIIF/`. The best source of English language information on the treatise and its author seems to be [11]; in Polish, it is [8]. To account for all the phonemes of Polish, Parkosz created some completely new letters. As his proposal did not catch on, the letters haven't became available in printer's fonts. He also assigned some different meanings to the variants of handwritten letters, which also were not used in print. The almost complete repertoire of Parkosz's letters is presented in Fig. 1.

Both the 1830 [2] and 1907 [10] editions of the treatise were typeset manually and undoubtedly special types had to be prepared for the missing characters. The 1985 edition [8] was typeset leaving empty space for Parkosz's letters. The letters were drawn by hand, cut into pieces and carefully pasted into the empty spaces.

Two amateurish attempts have been made (by the author and his student) to create a font for Parkosz's letters with FontForge. One was intended to reproduce the shape of the letters in the 1985 edition: `bitbucket.org/jsbien/parkosz-font-old`. Another was intended to reproduce the shape of the letters in the 1907 edition: `github.com/jsbien/parkosz-font`. However, their quality was not satisfactory, so for the electronic edition of the treatise [9] (see also `github.com/jsbien/Parkosz-traktat`) a transliteration system was designed. The rationale for the transliteration decisions were presented in Polish in [5] and summarized in English in [6]; see also Appendix A. The transliteration covers all the characters used by Parkosz, not just the newly created ones.

As the transliteration uses regular Unicode characters but changes their meaning, some metadata is needed to distinguish the normal text from the transliteration. It would be convenient to make the metadata unnecessary. Theoretically it is possible with so-called variation sequences, but they have to be officially registered by the Unicode Consortium, which makes this approach impractical, at least at the present stage.

In [4] a brute force approach was proposed. It assumed that after creating a TrueType/OpenType font with appropriate ligatures a little used combining character, namely U+20E8 combining triple underdot $\boxed{◌⃨}$, would serve as a kind of a private variant selector. At that time I was only vaguely aware of the usage of the so-called tag characters in Unicode.

In March 2022 Margaret Kibi (`marrus-sh`) proposed using tag characters instead of the variable sequences in the Junicode font.[1] The proposal was supported by several font users and accepted by the font author. I hope this approach will become a kind of a *de facto* standard, as it shares many advantages with TrueType/OpenType features (cf. [1, pp. 12–13]). To make a long story short, using those features preserves the properties of the base character, while a character in the Private Use Area has no properties.

This approach was also applied to Parkosz's letters as the implementation of my feature request[2] and this is the subject of the present paper. The characters accessed by variation sequences, tag characters or ligatures can have their own codepoints in the Private Use Area, but this is another topic.

[1] `github.com/psb1558/Junicode-font/discussions/122#discussioncomment-2416880`

[2] `github.com/psb1558/Junicode-New/issues/27`

doi.org/10.47397/tb/43-3/tb135bien-parkosz

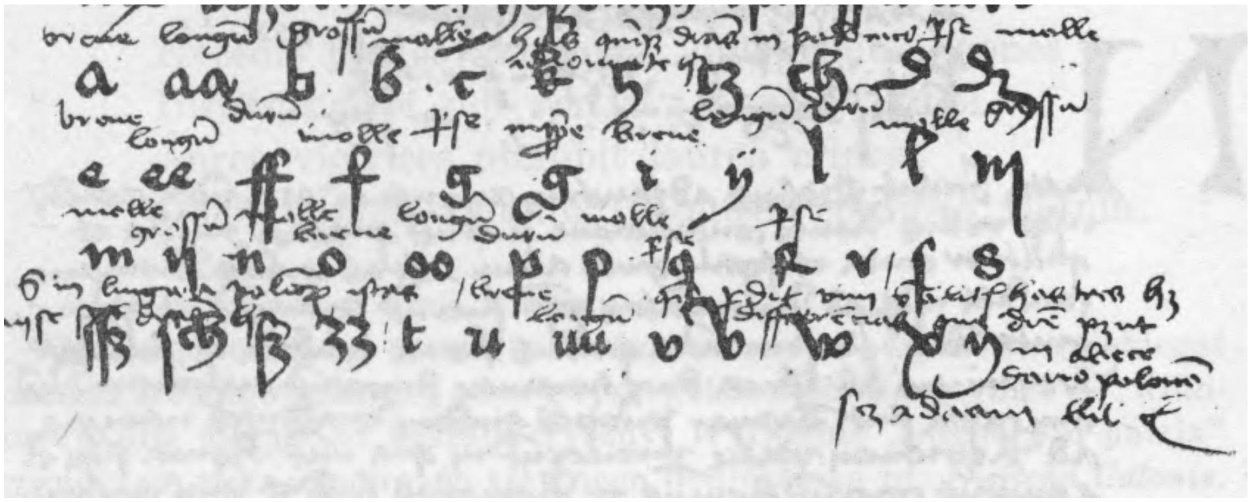


Figure 1: Parkosz's alphabet summary (with a few omissions)

Tag characters are in principle invisible, but for documentation purposes the tag sequences will be rendered here as, e.g., $\boxed{\text{p}}\boxed{\text{s}}$; *p* stands for *Parkosz*[3] and is always the very first tag, *s* refers to the shape and means *square*. Other secondary tags used are $\boxed{\text{r}}$ for *round*, $\boxed{\text{h}}$ for *hook* and $\boxed{\text{l}}$ for *loop*, $\boxed{\text{S}}$ for *slashed*, $\boxed{\text{b}}$ for *below*; $\boxed{\text{d}}$ means *descender* or *dot*.

In a XƎLaTeX source they can be written respectively as `\&\_\_p;`, `\&\_\_s;` etc. The same convention applies to XƎTeX and even to Microsoft Word and OpenOffice [1, p. 38].

The input can and should be simplified by using special Emacs input methods or equivalent macros in other tools. On the other hand usually just isolated words will be quoted, so providing the tag characters explicitly is not excessively tedious.

For the reader's convenience the layout of this paper is similar to [6] and some figures are repeated here. Please consult [6] for more details.

2 New letters

2.1 *b grossum*

Called also *b durum* and *b quadratum*. Some occurrences in the manuscript are presented in Fig. 2. It was transliterated as U+0180 LATIN SMALL LETTER B WITH STROKE ($\boxed{\text{ƀ}}$, Latin Extended-B block).

In Junicode the glyph is $\boxed{\text{ƀ}}$ and the input is **b**$\boxed{\text{p}}\boxed{\text{s}}$.

2.2 *b molle*

Called also *b rotundum*. Some occurrences in the manuscript are presented in Fig. 3.

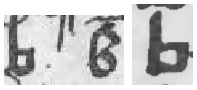


Figure 2: *b grossum* in the manuscript: p. [15] l. 28 (on the left), p. [8] l. 2

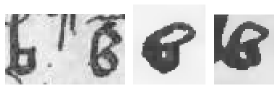


Figure 3: *b molle* in the manuscript: p. [15] l. 28 (on the right), p. [7] marginalia and l. 15.

It was transliterated as U+0253 LATIN SMALL LETTER B WITH HOOK ($\boxed{\text{ɓ}}$, IPA Extensions block).

In Junicode the glyph is $\boxed{\text{ɓ}}$ and the input is **b**$\boxed{\text{p}}\boxed{\text{l}}$.

2.3 *p durum*

Called also *p quadratum*. Some occurrences in the manuscript are presented in Fig. 4.

It was transliterated as U+1D7D LATIN SMALL LETTER P WITH STROKE ($\boxed{\text{ᵽ}}$, Latin Extended-C block).

In Junicode the glyph is $\boxed{\text{p}}$ and the input is **p**$\boxed{\text{p}}\boxed{\text{s}}$.

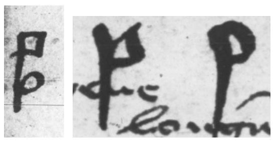


Figure 4: Letters *p durum* and *p molle* in the manuscript: p. [8] marginalia, p. [15] l. 32

[3] The tag is available in Junicode since build 1.052beta of August 25, 2022.

2.4 *p molle*

Some occurrences in the manuscript are also presented in Fig. 4.

The letter was transliterated as U+01A5 LATIN SMALL LETTER P WITH HOOK ($\boxed{\text{ƥ}}$, Latin Extended-B block).

In Junicode the glyph is $\boxed{\text{ρ}}$ and the input is **p**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{r}}{\text{TAG}}}$.

2.5 *l molle*

The letter was transliterated as U+026C LATIN SMALL LETTER L WITH BELT ($\boxed{\text{ɬ}}$, IPA Extensions block),

In Junicode the glyph is $\boxed{\text{ℓ}}$ and the input is **l**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{l}}{\text{TAG}}}$.

2.6 *durum v*

The letter was transliterated as U+028B LATIN SMALL LETTER V WITH HOOK ($\boxed{\text{ʋ}}$, IPA Extensions block).

In Junicode the glyph is $\boxed{\text{ν}}$ and the input is **v**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{h}}{\text{TAG}}}$.

3 Adapted letters

3.1 *g improprie*

This is the letter *g* as written by Italians (*unco retorto versus dexteram partem sicut scribunt ipsum Italici*).

The letter was transliterated as U+A77F LATIN SMALL LETTER TURNED INSULAR G ($\boxed{\text{ꝿ}}$, Latin Extended-D block).

In Junicode the glyph is $\boxed{\text{g}}$ and the input is **g**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{h}}{\text{TAG}}}$.

3.2 *grossum m*

This is the letter *m* as written at the end of words (*[...] spissum cum cauda, sicut in fine diccionum poni solet*). As discussed in [6, p. 48], the intended shape of the letter is not clear.

The letter was transliterated as U+0271 LATIN SMALL LETTER M WITH HOOK ($\boxed{\text{ɱ}}$, IPA Extensions block).

In Junicode the recommended input is **m**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{d}}{\text{TAG}}}$; it now renders $\boxed{\text{ꝳ}}$, i.e., the MUFI (Medieval Font Unicode Initiative[4]) Private Use Area character M+F223[5] LATIN SMALL LETTER M WITH RIGHT DESCENDER. This may change in the future.

3.3 *grossum n*

Similar to *grossum m*, this is the letter *n* as written at the end of words. As discussed in [6, pp. 48–49], the intended shape of the letter is not clear; see also Fig. 5.

The letter was transliterated as U+0272 LATIN SMALL LETTER N WITH LEFT HOOK ($\boxed{\text{ɲ}}$, IPA Extensions block).

In Junicode the recommended input is **n**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{d}}{\text{TAG}}}$; it now renders $\boxed{\text{ŋ}}$, i.e., the MUFI character M+F228 LATIN SMALL LETTER N WITH RIGHT DESCENDER. This may change in the future.

4 Special use letters

To this category belong the letters with the standard shape, but with a non-standard (from the contemporary point of view) phoneme assigned to them by Parkosz. They are: *f molle*, *g per se*, *l durum*, *m molle*, *n molle*.

All three editions used for these just the standard letters, which is quite confusing for present-day readers, even if they are scholars. Therefore the decision was made to transliterate them as, respectively:

- U+1E1F LATIN SMALL LETTER F WITH DOT ABOVE ($\boxed{\text{ḟ}}$),
- U+0121 LATIN SMALL LETTER G WITH DOT ABOVE ($\boxed{\text{ġ}}$),
- U+1E37 LATIN SMALL LETTER L WITH DOT BELOW ($\boxed{\text{ḷ}}$),
- U+1E43 LATIN SMALL LETTER M WITH DOT BELOW ($\boxed{\text{ṃ}}$),
- U+1E47 LATIN SMALL LETTER N WITH DOT BELOW ($\boxed{\text{ṇ}}$),

This part of my proposal can be considered an unnecessary complication, so should be treated as optional. Nevertheless in Junicode the characters can be entered as, respectively: **f**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{d}}{\text{TAG}}}$ ($\boxed{\text{ḟ}}$), **g**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{d}}{\text{TAG}}}$ ($\boxed{\text{ġ}}$), **l**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{b}}{\text{TAG}}}$ ($\boxed{\text{ḷ}}$), **m**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{b}}{\text{TAG}}}$ ($\boxed{\text{ṃ}}$) and **n**$\boxed{\overset{\text{p}}{\text{TAG}}}$$\boxed{\overset{\text{b}}{\text{TAG}}}$ ($\boxed{\text{ṇ}}$).

If used in colored text, it would be desirable to render the dot in a different color to make clear this is an artificial addition (an idea of Jakub Wilk formulated long ago in a different context). I am aware that because of technical difficulties this is at present rather a dream.

5 Regular letters

There is no typographical problem with the letters listed below. We give their names using original early Latin spelling (*breue* meaning *breve*, i.e., 'short').

They are: $\boxed{\text{a}}$ (*breue*), $\boxed{\text{c}}$, $\boxed{\text{d}}$ (*per se*), $\boxed{\text{e}}$ (*breue*), $\boxed{\text{i}}$ (*breue*), $\boxed{\text{ÿ}}$, $\boxed{\text{k}}$, $\boxed{\text{o}}$ (*breue*), $\boxed{\text{q}}$ (*per se*), $\boxed{\text{r}}$ and $\boxed{\text{R}}$ (*per se*, see also section 8), $\boxed{\text{ſ}}$ and $\boxed{\text{s}}$, $\boxed{\text{t}}$, $\boxed{\text{u}}$ (*breue*), $\boxed{\text{v}}$, $\boxed{\text{w}}$ and $\boxed{\text{x}}$.

As for an unnamed variant of the letter $\boxed{\text{c}}$, it is not obvious how to interpret it. I assumed this is U+00E7 LATIN SMALL LETTER C WITH CEDILLA ($\boxed{\text{ç}}$, Latin-1 Supplement block) and transcribed as such.

[4] `mufi.info`

[5] For referencing MUFI codepoints I advocate the use of the M+ prefix.

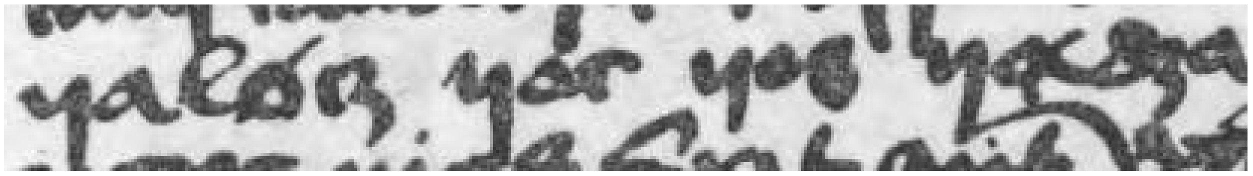


Figure 5: Grossum *n* in the manuscript at the beginning of words: p. [8] l. 14

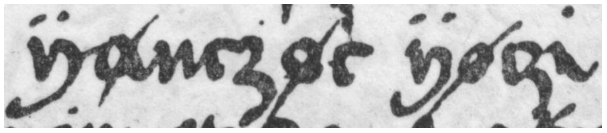


Figure 6: Nasal vowels in the manuscript

However in Junicode there is a special glyph for it, namely $\boxed{\text{ꞔ}}$ input as **c**$\boxed{\overset{\text{p}}{\scriptstyle\text{TAG}}}\boxed{\overset{\text{h}}{\scriptstyle\text{TAG}}}$.

In Junicode *long s* $\boxed{\text{ſ}}$ can be input as **s**$\boxed{\overset{\text{l}}{\scriptstyle\text{TAG}}}\boxed{\overset{\text{o}}{\scriptstyle\text{TAG}}}$.

6 Letters not listed in the alphabet summary

Most old Polish texts for the nasal vowel use U+A7C1 LATIN SMALL LETTER OLD POLISH O (ꟁ) introduced to Unicode in version 14.0 [7]. However in Parkosz's treatise the vowel has the shape of U+00F8 LATIN SMALL LETTER O WITH STROKE $\boxed{\text{ø}}$, cf. the second and the third stroked letter in Fig. 6, so it was used in the transliteration. However in Junicode it can be input as **ꟁ**$\boxed{\overset{\text{p}}{\scriptstyle\text{TAG}}}\boxed{\overset{\text{s}}{\scriptstyle\text{TAG}}}$ which makes its meaning clear.

The manuscript contains a single occurrence of U+2C65 LATIN SMALL LETTER A WITH STROKE (ⱥ), cf. the first stroked letter in Fig. 6, which could be a scribal mistake. Nevertheless it was transcribed just as $\boxed{\text{ⱥ}}$.

Letter $\boxed{\text{z}}$ was omitted in the alphabet, probably by mistake; in the transliteration it was used without change, which is a simplification, as it looks more like $\boxed{\text{ʒ}}$ U+0292 LATIN SMALL LETTER EZH.

The letter $\boxed{\text{h}}$ was also omitted probably by mistake.

7 Multigraphs

Some multigraphs in the manuscript are written as ligatures, but there is no need to represent them as such in the transliteration. On the other hand it would be desirable if the font rendered them as ligatures.

As it was mentioned earlier, some multigraphs consist of doubled vowels: *longum aa*, *longum ee*, *longum oo*, *longum uu*.

There is an open question whether the text would be more readable if *longum aa* was rendered as U+A733 LATIN SMALL LETTER AA (ꜳ, Latin Extended-D) and *longum oo* as U+A74F LATIN SMALL LETTER OO (ꝏ, also Latin Extended-D). For *longum ee* and *uu* the ligatures would have to be designed. The character M+E8C7 LATIN SMALL LIGATURE UU (ɯ) would be rather misleading.

The other multigraphs are: *ch* (taken over from Latin), *cz*, *molle dz*, *ſſz*, *ſch* and *ſz*, and *zz*.

8 Majuscules

Some of the examples are proper names, so they usually (not always) start with a majuscule (cf. the index in [9]).

The main Polish example is a verse, and the first words of some lines also start with a majuscule.

In consequence the transcription contained the following majuscules: $\boxed{\text{A}}$, $\boxed{\text{Ɓ}}$ (*B molle*, cf. Fig. 7), $\boxed{\text{C}}$, $\boxed{\text{Ġ}}$ (*G per se*), $\boxed{\text{I}}$, $\boxed{\text{K}}$, $\boxed{\text{N}}$, $\boxed{\text{Ᵽ}}$ (*P grossum*), $\boxed{\text{Ƥ}}$ (*P molle*), $\boxed{\text{Q}}$, $\boxed{\text{R}}$, $\boxed{\text{S}}$, $\boxed{\text{V}}$, $\boxed{\text{T}}$, $\boxed{\text{Z}}$.

In Junicode we have now $\boxed{\text{Ᵽ}}$ input as **P**$\boxed{\overset{\text{p}}{\scriptstyle\text{TAG}}}\boxed{\overset{\text{s}}{\scriptstyle\text{TAG}}}$ and $\boxed{\text{Ƥ}}$ input as **P**$\boxed{\overset{\text{p}}{\scriptstyle\text{TAG}}}\boxed{\overset{\text{r}}{\scriptstyle\text{TAG}}}$; $\boxed{\text{Ġ}}$ can be input as **G**$\boxed{\overset{\text{p}}{\scriptstyle\text{TAG}}}\boxed{\overset{\text{d}}{\scriptstyle\text{TAG}}}$.

It is perhaps worth noting that the upper case of *g per se* is $\boxed{\text{I}}$.

9 Concluding remarks

Adding Parkosz's letter to Junicode is in my opinion a large step forward in improving the editorial quality of publications concerning the treatise. Elsewhere I intend to present the difficulties caused by the lack of an appropriate font, including a misrepresentation of Parkosz's views.

It would be fun to have also a font simulating original handwriting, but the text would be unreadable for people without at least some knowledge of paleography; see, for example, the letters *d* and *x* shown in Fig. 8.

A An excerpt: a word list

According to [11], this is a mnemonic verse; in that paper you can also find the English translation.

A.1 Pure Unicode transliteration

Some minor mistakes in [9] corrected.

Adaaṃ biḷ bÿḷ caḷ kaal czas ꞔaḷo chood daaḷ dzaaḷ
eſz ﬃtaa fiꝯi i ġee ÿe ꝯhaaɲ kroɫ ḷis ɫis ɱikaa
ṃika ɲiſſki ṇiſki othooſz ꞗiġe piſchṇo qʋras roſſa

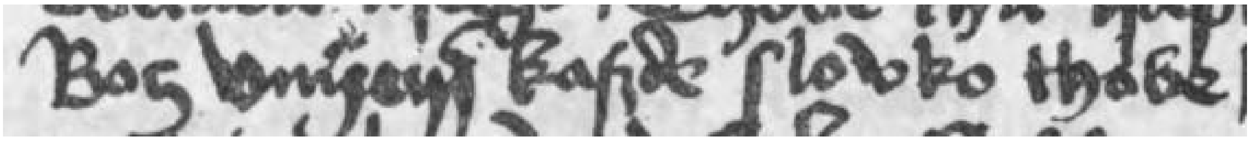


Figure 7: The manuscript p. 15 l. 24: Ɓoç ʋṇyeɱ kaſzde ſl̦oʋko thoɓe (transliteration)

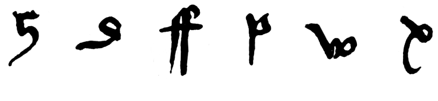


Figure 8: Outlines (by Szymon Pilas): ç, d, ff, p grossum, w, x

> rzøøſſa roſuuɱ. ſaaɱ ſchaad ſſzaadl̦ ſzak zzaraa
> Zaɱɲø to uṃee uuɲ vil̦a ʋiłaal̦ wſta xøødz
> ÿa̶ṇczøc ÿoczi ÿøøkaa

A.2 Junicode transliteration

The characters accessed with tag sequences are set in italics.

> Adaam bil bÿl cal kaal czas *çal̦*o chood daa dzaal
> e*ʃ*z ffitaa fi*g*i i ee ÿe *g*haa*n* kro*ł* *l̦*is *ł*is *ɱ*ikaa
> *ṃ*ika *ŋ*iſſki *ṇ*iſki othooſz *ρ*ige piſchno q*ʋ*ras
> roſſa rzøøſſa roſuu*ɱ*. ſaa*ṃ* ſchaad ſſzaad*l̦* ſzak
> zzaraa Za*ɱŋ*ø to u*ṃ*ee uu*ŋ* vi*l̦*a ʋi*ł*aa*l̦* wſta xøødz
> ÿa̶nczøc ÿoczi ÿøøkaa

A.3 A sample transcription

Various publications use various transcriptions to quote the treatise. Here is the one used in [11, pp. 125–126].

> Adam był bił cał kał; czas, ciało, chod dał dział; eż fyta figi i je je chan krol; łys lis myka, Mika nyski niski otoż pije pyszno kwas; rosa rząsa, rozum; sam szad siadł; żak ziara za mną; to umie un wiła; wylał w usta ksiądz, jęcząc jęczy, jąka.

References

[1] P. Baker. Junicode — the font for medievalists. specimens and user manual for version 2, 2022. `github.com/psb1558/Junicode-font/`

[2] J.S. Bandtkie, E. Raczyński, eds. *Jacobi Parkossii de Żorawice antiquissimus de orthographia polonica libellus*. Wilh. Deckeri et Societatis, Posnaniae, 1830. `www.wbc.poznan.pl/publication/115430`

[3] J.S. Bień. On standards for Computer Modern font extensions. *TUGboat* 11(2):175–183, 1990. `tug.org/TUGboat/tb11-2/tb28bien.pdf`

[4] J.S. Bień. Standard Unicode i dawny język polski. *Acta Poligraphica* 14:7–28, 2019. `10.5281/zenodo.4058701`

[5] J.S. Bień. Traktat Parkosza. Eksperymentalna edycja elektroniczna. *Poznańskie Studia Polonistyczne. Seria Językoznawcza* 26(1):27–69, 2019. `10.14746/pspsj.2019.26.1.2`

[6] J.S. Bień. Parkosz's treatise from a typographic point of view. *Scripta & e-Scripta* 22:37–57, 2022. `https://www.ceeol.com/search/article-detail?id=1058326`.
After fall 2023, also: `http://e-scripta.ilit.bas.bg/archives/year-2022/issue-22/yanush-s-bien-traktatt-na-parkosh-ot-tipografska-gledna-tochka`

[7] D. Bunčić. Proposal to include the letter 'Old Polish O' in ISO/IEC 10646 and the Unicode Standard. l2/21-039, 2021. `www.unicode.org/L2/L2021/21039-old-polish-o.pdf`

[8] M. Kucała. *Jakuba Parkosza Traktat o ortografii polskiej*. Państwowe Wydawnictwo Naukowe, 1985. `ebuw.uw.edu.pl/publication/220504`

[9] M. Kucała. Traktat o ortografii polskiej [Jakuba Parkosza]. Odczytanie, 2020. `10.5281/zenodo.3883863`

[10] J. Łoś. Jakóba syna Parkoszowego traktat o ortografii polskiej, 1907. `ebuw.uw.edu.pl/publication/238219`

[11] R. Wójcik, W. Wydra. Jakub Parkoszowic's Polish Mnemonic Verse about Polish Orthography from the 15th Century. In *Culture of Memory in East Central Europe in the Late Middle Ages and the Early Modern Period*, R. Wójcik, ed., no. 30 in Prace Biblioteki Uniwersyteckiej, pp. 119–127. Biblioteka Uniwersytecka, 2008. Ciążeń, March 12–14, 2008. `hdl.handle.net/10593/6037`

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

TeXShop, Version 5: HTML previews

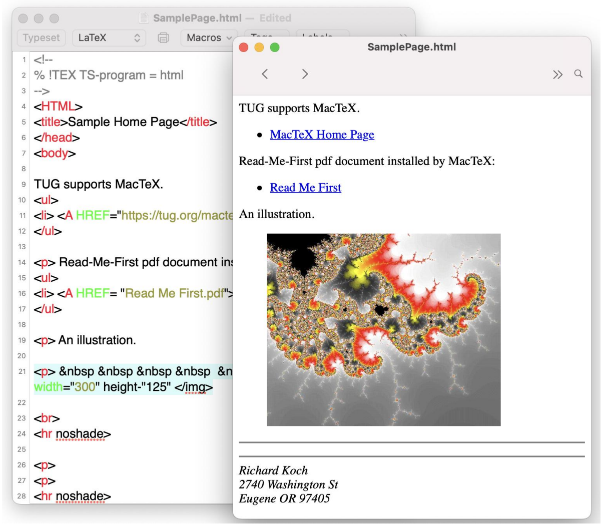
Richard Koch

Abstract

The Mac previewer TeXShop provides a preview window for each open document, showing pdf output. Version 5 of the program provides an extra preview window for html output. Therefore, TeXShop authors can use typesetting engines which convert source files to pdf, or html, or both. Examples include TEX4ht, which accepts LATEX source and outputs html, PreTEXt, which accepts xml source and outputs either pdf or html, and pure html source, which can be displayed as a live web page.

1 HTML source

Pictured below is the most straightforward example. The window on the left contains html source and the window on the right shows the resulting web page. Links to local or remote web pages work, as do links to pdf files, illustrations, and all other standard web content. After revising the source file, a user can type command-T, the standard TeXShop shortcut to typeset, and the web page will instantly update.



```
<!--
% !TEX TS-program = html
-->
<HTML>
<title>Sample Home Page</title>
</head>
<body>

TUG supports MacTeX.
<ul>
<li> <A HREF="https://tug.org/macte
</ul>

<p> Read-Me-First pdf document ins
<ul>
<li> <A HREF= "Read Me First.pdf">
</ul>

<p> An illustration.

<p> &nbsp &nbsp &nbsp &nbsp  &n
width="300" height-"125" </img>

<br>
<hr noshade>

<p>
<p>
<hr noshade>
```

TUG supports MacTeX.

- MacTeX Home Page

Read-Me-First pdf document installed by MacTeX:

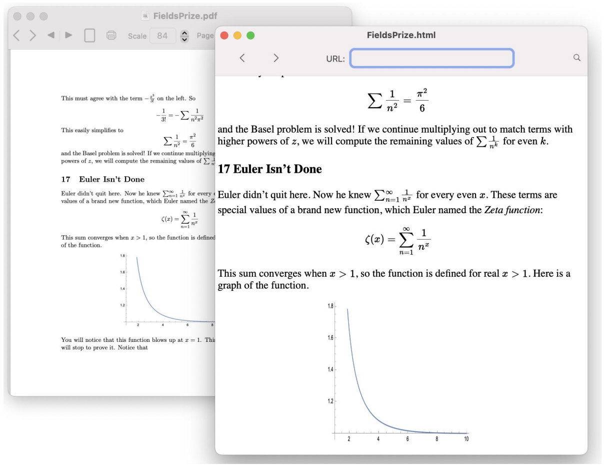
- Read Me First

An illustration.

Richard Koch
2740 Washington St
Eugene OR 97405

2 TEX4ht

Pictured next is an editing session using TEX4ht. To save space, the source window is not shown. The window on the left contains `pdflatex` output and the window on the right contains TEX4ht output. The TEX4ht engine, a very short shell script, typesets the source twice, once with `pdflatex` for the pdf preview and once with TEX4ht for the html preview. If the user edits the source and typesets again, both windows instantly update. In the illustration, both windows were resized to be very small. The resizing changed the magnification in the pdf window, but reflowed the text in the html window.



$$\sum \frac{1}{n^2} = \frac{\pi^2}{6}$$

and the Basel problem is solved! If we continue multiplying out to match terms with higher powers of z, we will compute the remaining values of $\sum \frac{1}{n^k}$ for even k.

17 Euler Isn't Done

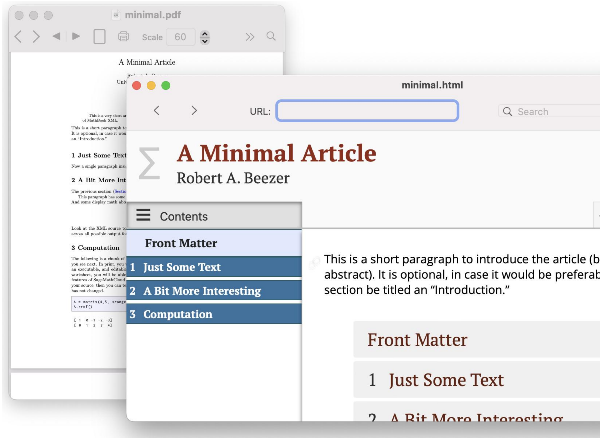
Euler didn't quit here. Now he knew $\sum_{n=1}^{\infty} \frac{1}{n^x}$ for every even x. These terms are special values of a brand new function, which Euler named the *Zeta function*:

$$\zeta(x) = \sum_{n=1}^{\infty} \frac{1}{n^x}$$

This sum converges when $x > 1$, so the function is defined for real $x > 1$. Here is a graph of the function.

3 PreTEXt

PreTEXt is a project managed by Robert Beezer at the University of Puget Sound. Pictured below is a typical editing session. The source window (not shown) contains xml source for document markup, but LATEX source for mathematics. Typesetting calls the `xsltproc` program twice, once to convert the source to pdf, and once to convert it to html. The pdf is shown on the left and the html is shown on the right. PreTEXt deliberately formats these outputs differently, so the pdf looks like standard TEX and the html looks like a web project. A key advantage of the PreTEXt project is support for many forms of user interactions in the html page.



A Minimal Article
Robert A. Beezer

This is a short paragraph to introduce the article (b abstract). It is optional, in case it would be preferab section be titled an "Introduction."

4 TeXShop engines

TeXShop typesetting engines are simple shell scripts stored in `~/Library/TeXShop/Engines`. A TeXShop user can navigate to this spot within the program, open files there, and edit them. Thus the user has complete control over the exact typesetting method used. In TeXShop 5.0, engine scripts can contain additional commands like

- `!TEX-pdfPreview`
- `!TEX-htmlPreview`
- `!TEX-bothPreview`

After typesetting, the first of these lines tells TeXShop to search for a file in the source directory with the same name as the source file and extension `.pdf`. If this file exists, it is opened in the pdf Preview window. The other commands work similarly.

5 Help recalling HTML commands

Several items in the TeXShop Help menu provide help recalling LaTeX commands. A new menu provides a list of html commands. The purpose of each command is listed, the appropriate tags are shown, and a short usage sample is provided. This help document can be typeset with command-T for easy viewing of examples explaining how to link to a pdf file, how to link to an external web page, how to display a movie, and how to display a YouTube video.

If the user needs a command not listed, they can search for it on Google. Then they can enter a new item in TeXShop Help, because the help file is editable. TeXShop will remember the edit, so the Help list will expand as the user adds items.

6 Different output for PDF and HTML

The PreTeXt system automatically adjusts input for display in pdf, html, and other formats. In TeX4ht, it is possible to manually write different source for pdf and html outputs. The recommended procedure is:

```
\ifx\HCode\undefined
  % source for pdf
\else
  <!-- source for html -->
\fi
```

Some web authors recommend a similar solution using "ifpdf", but that solution fails if the pdf output is typeset with XeLaTeX.

7 Purpose of new additions

Several talks at past TUG conferences predicted the demise of TeX in five or six years, because faculty members need to provide interactive source material. I thought these talks were ridiculous. Then Covid struck, and faculty colleagues had to switch to remote learning in a week. The pessimistic talks had a point.

In my "dream setup", a faculty member would write lecture notes in LaTeX or xml and these notes would be typeset by an engine which outputs both pdf and html. The pdf document would contain the course in final polished form, so students could understand the logical flow of the lectures. The html document would contain interactive material: movies, questions with student input, etc., so students would learn how mathematics is actually created.

I don't know what system faculty will use in the future, but I predict it will be able to output both pdf and html. This system might be PreTeXt, it might be TeX4ht, it might be software not yet invented. TeXShop is ready for all possibilities.

◇ Richard Koch
`koch (at) uoregon dot edu`
`https://pages.uoregon.edu/koch/`

Interactive content using TEX4ht

Richard Koch

Abstract

TEX4ht converts LATEX source into web pages. This article explains how to add interactive content to these pages, using TEX4ht and straightforward copying from web sources. The techniques should work on all computer platforms. Some refinements to the TEX4ht methods are also discussed.

1 Introduction

Let me begin with three vignettes.

I started attending TUG conferences in 2001, and along with expected talks there were a few surprises. In 2005, an expert from England predicted that TEX would survive for four more years and then be replaced. He was teaching in the Open University system where students work remotely, and he wanted to include interactive content in his lectures. I thought the talk was nonsense. Then COVID hit.

The 2004 Practical TEX conference was held at Fisherman's Wharf in San Francisco, and included a talk by Ernest Prabhakar, an Apple engineer. After that talk, Prabhakar met with Mac users and others including Hans Hagen, all sitting around a large conference table. Hans was trying to convince Apple to allow Java programs to run in their pdf viewer so interactive elements could be added. I sat next to Prabhakar and got to see how he operates. He was fully engaged in the conversation, but simultaneously he was surfing the web — the fastest surfer I have ever seen. Eventually he said to Hans, "It appears to me that you are the only one in the world writing Java in pdf files."

I'm one of those users who updates TEX Live daily while drinking my morning coffee. Sometime in 2022 I noticed that `tex4ht` was on every day's update list. So I wrote the TEX Live mailing list asking that this bug be fixed. To my surprise, I was told that the updates were genuine; Michal Hoftich, who maintains TEX4ht, makes updates almost daily.

2 PDF and HTML fifteen years later

The Fisherman's Wharf conference was 18 years ago, and some issues are clearer with the passage of time. Today every computer platform has excellent software to display pdf files, and every computer platform has an up-to-date web browser. It seems clear that pdf is the right format for static documents, and that html is the right format for documents with interactive content. Other formats may emerge, but that will only happen if an activity cannot be supported by pdf or html. (Although pdf has facilities for interactivity, they are rather infrequently used compared to interactive html.)

3 A TeXShop detour

I wrote TeXShop, a front end for TEX on the Macintosh. TeXShop is relevant here only because it explains how I was led to reexamine TEX4ht.

Typesetting in TeXShop is controlled by "engine files", small shell scripts that users can edit which call TEX binaries. After typesetting, an engine searches the source directory for a pdf file with the same name as the source and opens it in a pdf preview window if found. This August I added code which searches for an html file with the same name as the source, and opens it in an active web window if found. These windows are created using the Cocoa programming APIs, so they are part of TeXShop rather than external Mac applications like Preview and Safari.

With this change, it is easy to "typeset" html files. Similarly, it is easy to support PreTEXt, a project where authors write xml source and then convert the source to pdf, html, and other formats.

So it was natural to try TEX4ht, which accepts a LATEX source file and outputs html (among other things). TeXShop now has a typesetting engine which typesets the source twice, once with TEX4ht and once with pdflatex. The TEX4ht output is opened in a web viewer and the pdf output is opened in a pdf preview.

I had seen demonstrations of TEX4ht given by Eitan Gurari, the original author of TEX4ht. Indeed at that 2004 conference at Fisherman's Wharf, Prabhakar's talk was immediately followed by a talk by Gurari on TEX4ht. At the time, TEX4ht was outputting mathematics using pictures, and the results were a little crude.

In the years since then, MathML was invented, and then MathJax was created and provided beautiful rendering of MathML code. TEX4ht adopted these technologies.

I selected a 20-page set of lecture notes, with extensive mathematical equations and many illustrations. The document used `hyperref`, `amsmath`, and other packages. I typeset it with TEX4ht, producing html. Typesetting was fast — and the html output was amazing! The mathematical equations were crisp and clear, the illustrations were fine; to tell the truth, I doubted that I was seeing html. As a test, I resized both windows. The text in the pdf window shrank since the pdf had been configured to "fit in window". The text in the html window reflowed.

4 Interactive content

TEX4ht therefore allows you to convert old and new LATEX static documents into web documents. But can you add interactive content to these documents? Yes, as this article will demonstrate.

Select an old document you have lying around the house. You'll be able to add interaction to it by the end of the next two sections. Don't typeset immediately because a couple of steps are needed. Both are given in these two sections.

First we need a method to write source code which will only appear in the html version of the document. The following code does the trick:

```
\ifx\HCode\undefined
  % source for pdf document
\else
  <!-- source for html document -->
\fi
```

The `\HCode` tested here is a command that appears only in TEX4ht. Some web documents recommend the `ifpdf` package, but that fails when typesetting with XƎTEX.

Next, we need to switch from writing LATEX code to writing html code which TEX4ht will insert verbatim into the final document without processing. The following code suffices:

```
\ifx\HCode\undefined
  % source for pdf document
\else
  Initial words for html document.
  \begin{html}
     <!-- direct html input -->
  \end{html}
\fi
```

Finally we need something interactive. We'll use a piece of SageMath code, which is explained in a later section. Putting all this together, add the following lines to your document, creating a new section in the web version.

```
\ifx\HCode\undefined
\else
 \section{An Experiment}
 \begin{html}
  <div class="compute">
  <script type="text/x-sage">
    plot(sin(x), (x, 0, 2*pi))
  </script></div>
 \end{html}
\fi
```

Running this, we discover a minor problem. TEX4ht does not understand the command `\begin{html}`, and your web browser does not understand the lines calling Sage.

5 The header

To solve the problem, we must add the following header immediately after `\begin{document}`, before any other code is inserted. (The columns in *TUGboat* are narrow; the long `sagemath.org` url below needs to be on one line, and other source lines would usually be combined. Also, the source for this article is available from the article's web page.)

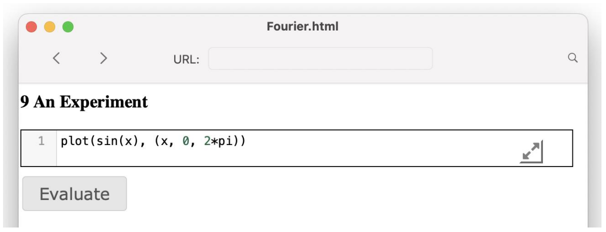
```
\ifx\HCode\undefined
\else
 % declare environment html:
 \ScriptEnv{html}
 {\ifvmode\IgnorePar\fi\EndP
   \NoFonts\hfill\break}
 {\EndNoFonts}
\fi

\ifx\HCode\undefined
\else
 % following url needs to be on one line, sorry:
 \begin{html}
  <script src="https://sagecell.sagemath.org/
static/embedded_sagecell.js">
  </script>
  <script>
  // Make div with id `mycell' being a Sage cell
  sagecell.makeSagecell({
    inputLocation:  '#mycell',
    template:  sagecell.templates.minimal,
    evalButtonText: 'Activate'});
  // Make div with class `compute' a Sage cell
  sagecell.makeSagecell({
    inputLocation: 'div.compute',
    evalButtonText: 'Evaluate'});
  </script>
 \end{html}
\fi
```

This header is divided into two parts, each preceded by an `\HCode` test so it is only active when typeset by TEX4ht. The first defines `\begin{html}` for TEX4ht. The second defines the Sage commands for the browser. Notice that the second command is also preceded by `\begin{html}` and thus is inserted directly into the final html document.

Now your source document has everything required for interaction, so go ahead and typeset it with TEX4ht. The recommended way to typeset is

```
make4ht sourcefile.tex "mathjax"
```

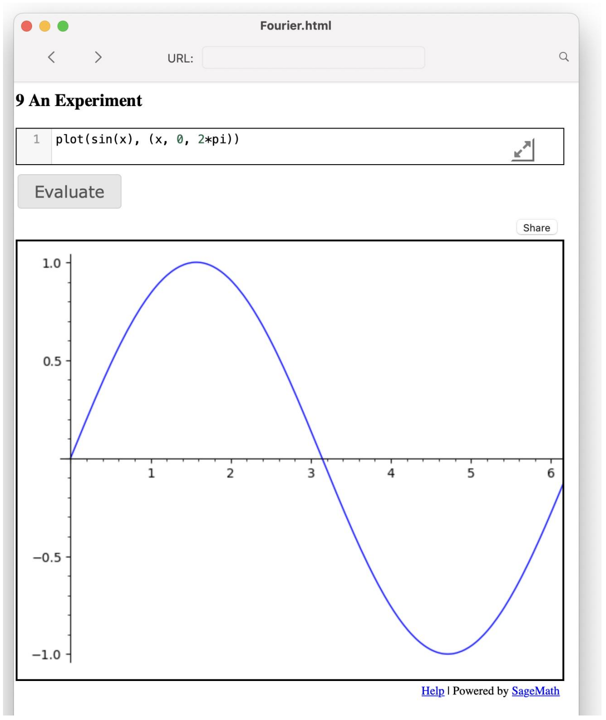
In the pdf version of the document, nothing changed. The html version will contain an additional section pictured below. Notice the button labeled "Evaluate" (fig. 1). When this button is pushed, the display changes to the form shown on the second image (fig. 2).



9 An Experiment

```
plot(sin(x), (x, 0, 2*pi))
```

Figure 1: Sage Evaluate button and editable function.



9 An Experiment

```
plot(sin(x), (x, 0, 2*pi))
```

Figure 2: Original plot results.

But there's more. The SageMath code shown in both images is editable. If this entry is changed to

```
plot(log(x), (x, .1, 10))
```

a graph of the logarithm is plotted, and if it is changed to

```
plot(sin(x) + cos(3*x)/2, (x, 0, 2*pi))
```

an alternate periodic function is plotted. Therefore the four lines of code we added for Sage did not just provide a plot of the sine function. It provided a plotting machine which readers can use to plot any function!

6 SageMath

SageMath is an open source alternative to the computer algebra systems Magma, Maple, Mathematica, and MATLAB. The project was created by William Stein, a mathematician at the University of Washington, and first released on February 24, 2005. See sagemath.org and wiki.sagemath.org. Sage is mostly written in Python, but it integrates many previous open source projects written in C, Lisp, and Fortran. Among these are Gap, Macaulay, Maxima, Octave, and R. The program has been used for serious research on elliptic curves, finite groups, and many other areas, and has an active support group.

Although the Sage web site has install packages for major computer platforms, our use of Sage does not depend on installing SageMath, either for the author or for the reader on the web. Instead, Sage maintains a server which can run Sage over the web. See https://sagecell.sagemath.org and other links from that page for details.

Our previous Sage example contains a single line to plot a function. However, that line can be replaced by an arbitrary Sage program, which can be several pages long. We list several examples. All of these examples come from web pages at the Sage site; I have just copied and pasted code by others.

Here are two examples of calls to Sage:

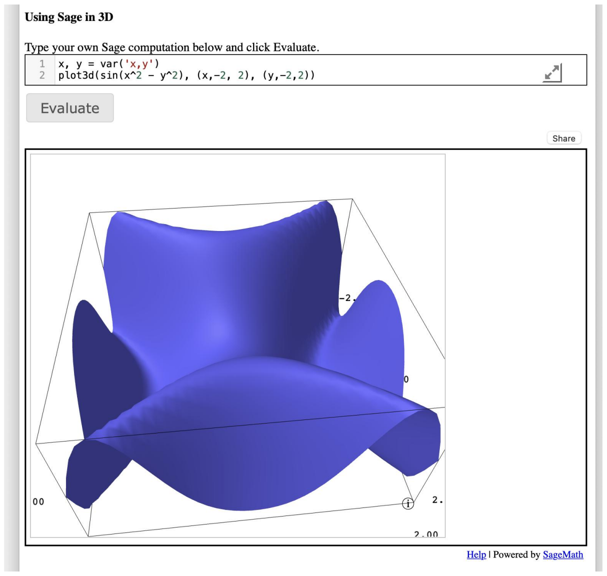
```
\begin{html}
<div class="compute">
<script type="text/x-sage">
x, y = var('x,y')
plot3d(sin(x^2 - y^2), (x,-2, 2), (y,-2,2))
</script></div>
\end{html}
```

and

```
\begin{html}
<div class="compute">
<script type="text/x-sage">
u,v = var('u,v')
fx = (3+sin(v)+cos(u))*cos(2*v)
fy = (3+sin(v)+cos(u))*sin(2*v)
fz = sin(u)+2*cos(v)
parametric_plot3d([fx, fy, fz], (u, 0, 2*pi),
       (v, 0, 2*pi), frame=False, color="red")
</script></div>
\end{html}
```

The output from executing these commands is shown on the next page (fig. 3). However, this (pdf) article doesn't show the most amazing thing. If these objects are grabbed with the mouse, they rotate and magnify instantly in real time.

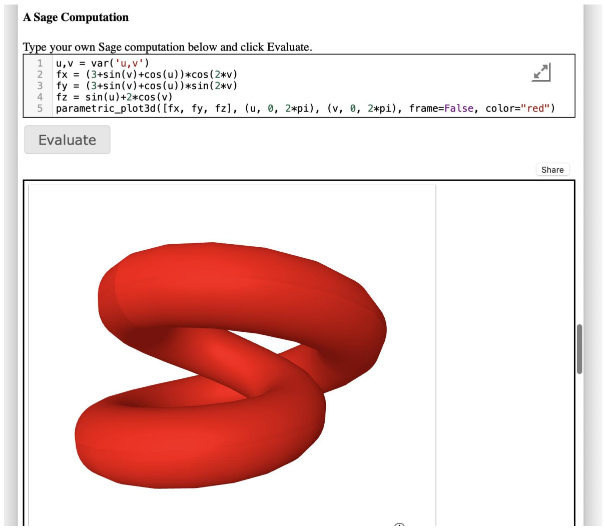
For the mathematically inclined, I'll show two more examples on the next page, without giving the Sage code, which can be found on various Sage web sites. Figure 4 illustrates numerical integration. The function can be set by the reader, the number of division points can be set, and the algorithm determining the top of each rectangle can use the value of the function at the left, right, or middle, or the maximum or minimum value. Since Sage can integrate symbolically, the exact value of the integral



Using Sage in 3D

Type your own Sage computation below and click Evaluate.

```
x, y = var('x,y')
plot3d(sin(x^2 - y^2), (x,-2, 2), (y,-2,2))
```



A Sage Computation

Type your own Sage computation below and click Evaluate.

```
u,v = var('u,v')
fx = (3+sin(v)+cos(u))*cos(2*v)
fy = (3+sin(v)+cos(u))*sin(2*v)
fz = sin(u)+2*cos(v)
parametric_plot3d([fx, fy, fz], (u, 0, 2*pi), (v, 0, 2*pi), frame=False, color="red")
```

Figure 3: Two fancy plots, which in html output can be transformed in real time.

is shown at the bottom, and finally the numerical approximation is computed and shown.

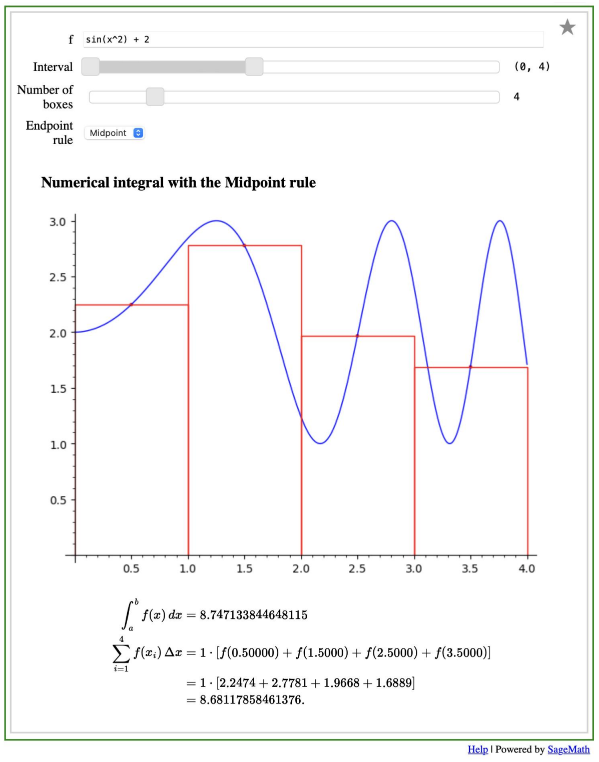
In figure 5 the Taylor series of a function selected by the user is computed, and both the function and its approximation are plotted.

7 YouTube videos

Did you know that if you right-click while playing a YouTube video, a contextual menu appears allowing you to “copy embed code”, which can be pasted into a web page?

I found a lecture by John Maynard, one of the four Field’s Prize winners at the International Congress of Mathematicians for 2022. It is fun to watch this video for the depth and clarity of his mathematics. I confess that I also watched because Maynard is left-handed and we lefties need to stick together. I don’t understand a word of the code which YouTube provided when I clicked, but I added it to a TEX4ht source page and it worked. Figure 7 shows a frame of the video, and here is some of the source code, as copied from YouTube:

```
\begin{html}
```



f sin(x^2) + 2

Interval (0, 4)

Number of boxes 4

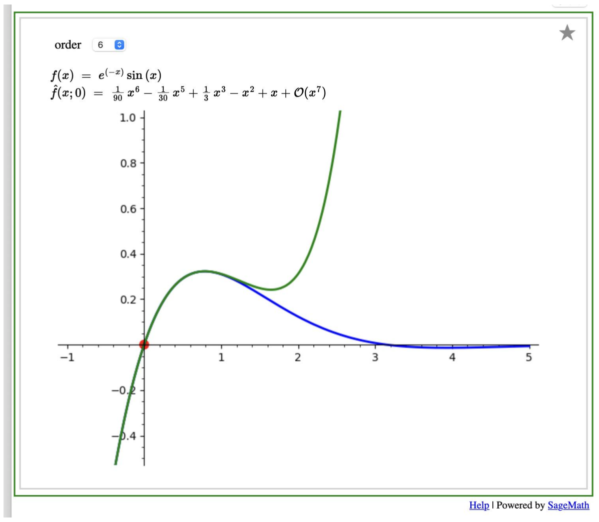
Endpoint rule Midpoint

Numerical integral with the Midpoint rule

$$\int_a^b f(x)\,dx = 8.747133844648115$$

$$\sum_{i=1}^{4} f(x_i)\,\Delta x = 1\cdot[f(0.50000)+f(1.5000)+f(2.5000)+f(3.5000)]$$
$$= 1\cdot[2.2474+2.7781+1.9668+1.6889]$$
$$= 8.68117858461376.$$

Figure 4: Numerical integration example with Sage.



order 6

$$f(x) = e^{(-x)}\sin(x)$$
$$\hat{f}(x;0) = \tfrac{1}{90}x^6 - \tfrac{1}{30}x^5 + \tfrac{1}{3}x^3 - x^2 + x + \mathcal{O}(x^7)$$

Figure 5: Taylor series of a user-selected function.

13. Large gaps between primes in subsets - James Maynard (University of Oxford) [2...
Watch on YouTube

Figure 6: Frame from YouTube video, playable live in html output.

```
<iframe width="928" height="522"
 src="https://www.youtube.com/embed/kQqBeuk_xQw"
 title="13. Large gaps between primes ..."
 frameborder="0"
 allow="accelerometer; autoplay; ..."
 allowfullscreen></iframe>
\end{html}
```

8 Mathematics

Often authors ask a question of readers and provide a multiple choice answer. If the reader answers correctly, they are told to go to the next section; otherwise new text appears explaining why their answer was incorrect.

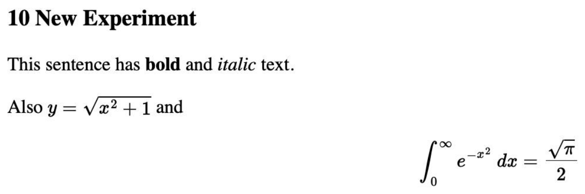
In a mathematical text, both the question and the various answers will likely contain mathematical formulas. But recall that the interactive material is being written in html and inserted directly in the final document without processing. Are authors expected to write the mathematics in MathML? If they write in LaTeX, TeX4ht cannot convert the code to MathML because it doesn't touch the author's html blocks.

The happy answer is that, with MathJax, authors *can* directly write LaTeX math, even inside the verbatim `{html}` environment we've defined (because MathJax recognizes the math). There is one caveat: normally, inline math can be specified by either a pair of `$` signs or a `\(` and `\)` pair, but inside `{html}` and when using MathJax, `\(...\)` must be used (or extra MathJax configuration specified). `$...$` works fine with MathJax outside of our `{html}` environment.

Display math can be defined by a pair of `$$` signs or a `\[` and `\]` pair; both these forms work inside `{html}`, with MathJax and otherwise.

```
\ifx\HCode\undefined
\else
 \section{New Experiment}
 \begin{html}
   <p>This sentence has <b>bold</b>
   and <i>italic</i> text.</p>
   <p>Also math: \(y = \sqrt{x^2 + 1}\) and
   $$\int_0^\infty e^{-x^2} \ dx =
   {{\sqrt{\pi}} \over 2}$$</p>
\end{html}
\fi
```

Typeset and you will see the output below (right margin has been truncated).



10 New Experiment

This sentence has **bold** and *italic* text.

Also $y = \sqrt{x^2 + 1}$ and

$$\int_0^\infty e^{-x^2}\ dx = \frac{\sqrt{\pi}}{2}$$

But how is this possible, since source inside an "html pair" is inserted directly in the output without processing?

9 Calling TeX4ht

Originally TeX4ht output small pictures for inline and displayed mathematics. Eitan Gurari unexpectedly died in 2009, and TUG paid him the ultimate compliment by keeping his program alive. Now it is actively maintained by Michal Hoftich.

Due to new developments in MathML and MathJax, there are many ways to call TeX4ht when it is asked to typeset. Let us concentrate on the three most important methods.

Calling TeX4ht using the call

```
make4ht source.tex "mathml"
```

causes TeX4ht to insert MathML code for inline and display equations. This MathML is then rendered by the browser.

Calling TeX4ht using the call

```
make4ht source.tex "mathml,mathjax"
```

causes TeX4ht to insert MathML code for inline and display equations, but call MathJax to render the resulting code.

Calling TeX4ht using the call

```
make4ht source.tex "mathjax"
```

causes TeX4ht to insert LaTeX code for inline and display equations, and call MathJax to render the resulting code.

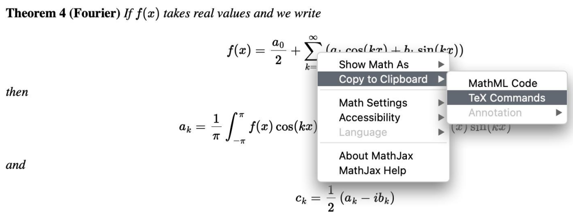
Note that MathJax can render both MathML and LaTeX code when it discovers equations in an html document.

On my computer, mathematical rendering using the first method is not as clear as rendering with the other two methods. Integral signs are too small and there are other minor flaws. The first and third methods understand LaTeX input for interactive content, but the second does not. These experiments suggest that the third method is the most desirable for interactive code.

My initial experiments did not go well with the third method. Inline equations were fine, but displayed equations were rendered with static images. Then one day I tried the alternate `\[` notation rather than `$$` and everything worked. I reported this to Michal, and the *very next day* he fixed TeX4ht so both notations are rendered with MathJax. (The LaTeX developers do recommend `\[...]`, by the way.) Please update your TeX Live distribution and typeset using the third method.

10 A MathJax perk

By now, perhaps you have typeset your own document with TeX4ht and MathJax. Select an equation and right click on it. A contextual menu opens offering to copy the equation to the clipboard as either "MathML" or "TeX Commands". Here's a picture:



Theorem 4 (Fourier) *If $f(x)$ takes real values and we write*

$$f(x) = \frac{a_0}{2} + \sum_{k=}^{\infty} (a_k \cos(kx) + b_k \sin(kx))$$

then

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(kx) \qquad (x) \sin(kx)$$

and

$$c_k = \frac{1}{2} (a_k - ib_k)$$

Select "TeX Commands", copy, and paste somewhere else. You will obtain the LaTeX code for the equation. This code can be copied into any other LaTeX source document.

This remarkably useful feature comes from MathJax and is not available if you call TeX4ht using the first method. Moreover, the menu will offer MathML code, but not LaTeX code, if you call TeX4ht using the second method. But the third method of calling TeX4ht gives LaTeX code.

11 Installing documents on the server

Suppose you typeset a document named `Sample` with TeX4ht and produce `Sample.html`. How should this file be put on a server? The answer is tricky because `Sample.html` itself will not contain any images, so any needed image files must be provided separately. Moreover, TeX4ht generates a support file `Sample.css`, which is also required.

Thus it is convenient to put all illustrations in a folder, named (say) `Graphics`, and refer to these illustrations in the LaTeX source using the pattern `Graphics/plot1` (LaTeX will automatically look for usual image extensions, using whichever is found). Then the web server should contain `Sample.html`, `Sample.css`, and the `Graphics` folder.

12 Using the work of other people

Nothing in this document comes from me. When I discovered that TeX4ht produces completely acceptable web pages, I wondered if it would accept html code and send it unmodified to the html document. I asked Karl Berry, who thought it was possible and asked Michal Hoftich. Michal sent the method described here, but I didn't believe it was sufficiently general. So I started writing a sample document showing that the method could not display math, or handle YouTube videos, or accept Sage code. My sample simply proved the opposite.

I do not know a single MathML tag. I knew the American Mathematical Society recommended MathJax, but didn't know why. I don't understand how these technologies work.

Several years ago I downloaded Sage. But I didn't know that web pages could access a server so students who had never installed Sage could still read web pages with Sage content. When I realized that, I used Sage to graph simple functions. When it displayed a 3D graph and let me rotate it interactively, I almost fell off my chair.

It is strange that I had to learn these lessons over again, because LaTeX is a crucial tool for me and yet I have never read *The TeXbook*; TeX macros are crucial for my life and yet I don't know how to write a macro. We can do things in our lives because of the independent work of thousands of people.

13 PDF and HTML in mathematics

When I was a college sophomore, I took an abstract algebra course from W. Wistar Comfort. His lectures were crystal clear; you could copy the board, read the notes at home, and see every step in its proper logical order.

Later I took courses with a more rough and tumble atmosphere; the instructor seemed to be inventing right in front of our eyes, and sections of the board would be crossed out when a better idea presented itself.

Both lecture styles worked, showing the dual nature of mathematics. To me, pdf is for the final crystalline form of mathematics, and html is for the rough and tumble way it is invented. Euclid is pdf, but Legendre is html, and Euler is both.

◇ Richard Koch
`koch (at) math dot uoregon dot edu`
`http://pages.uoregon.edu/koch/`

A Refinements for TEX4ht

(Everything in this section came from Michal Hoftich, who we asked to review the above.)

A.1 `\ifdefined\HCode`

The main article uses

```
\ifx\HCode\undefined\else ... \fi
```

to insert material only when processing under TEX4ht. This is fine, and is the general form. But when only the html output needs the extra attention, it can be simplified to:

```
\ifdefined\HCode ... \fi
```

(By the way, `\ifdefined` is an ε-TEX primitive; LATEX has required ε-TEX, and some primitives beyond ε-TEX, for years now.)

A.2 `\NewDocumentEnvironment{html}`

The main article uses the `{html}` environment inside `\HCode` conditionals, so that only TEX4ht sees it. This is fine, but it is arguably nicer to define the `{html}` environment in all cases, and make it a no-op when being processed for pdf (or dvi, but we won't keep mentioning that).

Also, we may as well define an analogous environment for material that should only be processed in the pdf case.

This can most easily be done using the relatively recent (2020) `\NewDocumentEnvironment` command. The following two definitions in the preamble define an `{html}` environment to ignore its contents (since normally we are running LATEX, not TEX4ht), and the `{pdfenv}` environment to typeset its contents (for the same reason):

```
\documentclass{article}
\NewDocumentEnvironment{html}{+b}{}{}
\NewDocumentEnvironment{pdfenv}{}{}{}
```

Then, in a configuration file for TEX4ht (see next section), we reverse the definitions so that `{html}` is active and `{pdfenv}` is a no-op:

```
% (in a configuration file, see below)
\ScriptEnv{html}
 {\ifvmode\IgnorePar\fi\EndP\NoFonts\hfill\break}
 {\EndNoFonts}
\RenewDocumentEnvironment{pdfenv}{+b}{}{}
```

Then the environments can be used without any conditionals. As a side benefit, the environments can be nested. For example:

```
\begin{document}
...
\begin{html}
<p>This is output only in HTML, but can include
   LaTeX math: \( a=b^2 \).</p>
\end{html}
\begin{pdfenv}
Nested \LaTeX\ not in the HTML output.
\end{pdfenv}

\begin{html}
<p>Then we can have more HTML.</p>
\end{html}
```

A.2.1 `\NewDocumentEnvironment` explanations

You may be wondering what the `+b` means in the `\NewDocumentEnvironment` call. If you're not wondering, skip this section.

The environment name (e.g., `html`) is the first argument to `\NewDocumentEnvironment`. The second argument, with the `+b`, defines how arguments should be handled. The third and fourth arguments, empty for us, define the code which is run at the beginning and end of the environment, respectively.

The `b` argument specification says to pass the body of the environment as argument `#2` to the code blocks. (`#1` is for the optional argument, which we don't use.) The `+` specifier allows multiple paragraphs within the environment body.

Since we don't specify any code to run, nothing is done with the environment body, so it is effectively discarded. On the other hand, when the argument specification is empty, the environment body is processed normally.

Many powerful argument specifiers are available, and they can be used when defining either environments or commands. See the LATEX `usrguide3` document for details.

A.3 TEX4ht configuration files

TEX4ht supports configuration files, which are a convenient way to specify document-wide settings. The environment redefinitions shown above are one example. Here is another example, moving the Sage specifications to the html page header (via `@HEAD`):

```
\Configure{@HEAD}{%
   <script src="https://sagecell...\Hnewline
   ...
   % must escape the # character:\Hnewline
   inputLocation:  '\#mycell',\Hnewline
   ...
   </script>\Hnewline}
```

Because the configuration file is ultimately TEX code, it is necessary to escape `#` with a backslash, and explicitly insert newlines in the output with `\Hnewline`, as shown.

If the configuration file is saved as `conf4ht.cfg` (the name can be anything), the `make4ht` call becomes:

```
make4ht --config conf4ht.cfg source.tex "mathjax"
```

What's new in TEX4ht: 2022

Michal Hoftich

Abstract

This article provides an overview of the recent development of TEX4ht, LATEX to XML converter, and `make4ht`, the build system that carries out this conversion.

1 Introduction

Richard Koch wrote an article on interactive documents produced using TEX4ht in this issue of *TUGboat*. He and Karl Berry asked if I would be able to provide additional tips on the usage of TEX4ht and also to summarize recent changes in the system.

You can find the basic summary of the basic features of TEX4ht in my previous article [1]. I will focus on new features and changes in this article.

2 Changes and new features in `make4ht`

There are some substantial changes in the `make4ht` build system. These are the most important:

2.1 Terminal output

Originally, `make4ht` showed the full terminal output of TEX and all the commands it called during the conversion process. This resulted in a huge amount of information printed on the terminal. It also used the default behavior of LATEX, so the compilation was stopped for every error, waiting for the user action.

The new default behavior is to run the compilation in `\nonstopmode`, with most terminal output suppressed. Only errors and warnings are shown.

You can change the output method using a new command line option `--loglevel`, or `-a` in the short form. Each log level prints messages of the current level and all higher levels. It supports the following levels:

error print only error messages.

warning show `make4ht` warnings, for example, from HTML postprocessing filters.

status this is the default level.

info print all `make4ht` messages, but suppress the output from commands.

debug this level is the original default, printing all output from TEX and all other executed programs, and it also stops on compilation errors.

2.2 Input redirection

`make4ht` now supports shell input redirection, which means that it can process the output of other commands without the need to use temporary files. You need to pass `-` as the filename, and also set the output filename using the `--jobname` or `-j` option:

```
$ python generatetex.py | make4ht -j foo -
```

2.3 Conversion of additional markup languages

In addition to LATEX and plain TEX, `make4ht` supports some additional markup languages, thanks to the `preprocess_input` extension. It detects the markup used using the file extension, so it is necessary to name the file accordingly. It preprocesses the input using Pandoc or R with the Knitr library, which needs to be installed on your system.

Here's the list of supported file extensions:

`.rtex`	LATEX with R code chunks
`.rnw`	LATEX with Sweave code chunks
`.rmd`	RMarkdown
`.rrst`	R + reStructuredText
`.md`	Markdown
`.rst`	reStructuredText

For example, the following LATEX document contains R commands, so we name it (say) `x.rtex`:

```
\documentclass{article}
\begin{document}
You can have R commands in your \LaTeX{}
document. They will be processed and
their output will be typeset:
<<>>=
# Create a sequence of numbers
X = 2:10
# Summary of basic statistical measures
summary(X)
@
\end{document}
```

You can compile it with the following command, which loads the `preprocess_input` extension:

```
$ make4ht -f html5+preprocess_input x.rtex
```

2.4 New commands available in build files

With `make4ht`, you can use Lua build files to call additional commands, such as indexing and bibliography processors. Built-in commands are provided for Biber, BIBTEX, Makeindex, Xindy, Xindex and PythonTeX. They take care of the special settings necessary to work correctly with TEX4ht.

As an example, the following document produces an index with links that point to the places where `\index` is used:

```
\documentclass{article}
\usepackage{makeidx}
\makeindex
\begin{document}
Hello\index{hello} world\index{world}
\printindex
\end{document}
```

doi.org/10.47397/tb/43-3/tb135hoftich-make4ht

A build file, say `build.lua`, that uses Makeindex as an indexing processor could look like this:

```
Make:htlatex {}
Make:makeindex {}
Make:htlatex{}
```

The command `Make:htlatex` compiles the document using LaTeX with the TeX4ht package automatically loaded, `Make:makeindex` calls Makeindex and the final `Make:htlatex` compiles the document with the index included. Note that instead of page numbers, the numbers in the index are numbered consecutively for each `\index` command. Due to that, we can point every index entry back to the original location.

To use a build file, use the `-e` command line option:

```
$ make4ht -e build.lua foo.tex
```

3 Documentation and server side compilation

We have made progress in writing new TeX4ht documentation. It contains chapters on available configuration commands and command line options, and also a how-to guide with common tasks. Developer information for package writers is also included. It is available here:

www.kodymirus.cz/tex4ht-doc

It also describes an important development, the usage of server-side compilation. Thanks to Github Actions, documentation is automatically generated from LaTeX sources every time we update them. We don't need to upload generated HTML files to a web server, everything is handled automatically by Github Actions. In the background, the Docker container for TeX Live is used. It enables us to call any command available in TeX Live, including `make4ht`. A similar service is also provided by GitLab and other source code hosting platforms.

This method has also been used for the conversion of Overleaf projects linked to Github repositories, the HTML version of `make4ht` documentation, and even a simple blog:

www.kodymirus.cz/testblog/

4 JATS format support

We recently added support for the JATS XML format, which is intended for scientific article authoring. This is an important development, as this format is required by many publishers for article archiving or further processing.

It is also the first output format for TeX4ht that I personally created. The specification is quite strict on the structure of the document, which is often inconsistent with the free document structure used in LaTeX. `make4ht` postprocessing using LuaXML is heavily used to produce the correct structure.

The support is still fairly basic, so user feedback and bug reports are appreciated. The basic invocation:

```
$ make4ht -f jats foo.tex
```

5 MathJax configuration

We continue to extend support for MathJax, which can be used to render math in converted documents. The resulting document typically looks much better than documents converted using the default TeX4ht method, which uses a mix of images and HTML formatting. It is also better for accessibility, as MathJax can support screen readers, for example.

One pitfall is that MathJax does not support custom commands out of the box. It needs to be given special configuration that declares these commands. Here is an example of such a configuration file for a hypothetical macro `\foo`:

```
\Preamble{xhtml,mathjax}
\Configure{MathJaxConfig}{{
tex: {
 \detokenize{%
  macros: {
    foo: "\\mathrm{foo}",
  }
 }
}
}}
\begin{document}
\EndPreamble
```

The `\Configure{MathJaxConfig}` command here is given JavaScript code that configures MathJax. The `macros` table, which needs to be located inside the table `tex`, can contain user macros. The `\detokenize` (ε-TeX) command is used to prevent problems with backslash characters, which need to be doubled. In the example, we define the `\foo` macro, which prints the word "foo" in roman font.

References

[1] M. Hoftich. TeX4ht: LaTeX to Web publishing. *TUGboat* 40(1):76–81, 2019. tug.org/TUGboat/tb40-1/tb124hoftich-make4ht.pdf

◇ Michal Hoftich
michal dot h21 (at) gmail dot com
https://tug.org/tex4ht/

Adding XMP metadata in LaTeX

Ulrike Fischer, Frank Mittelbach

Abstract

One task of the "LaTeX Tagged PDF Project" [6] is to evaluate existing solutions to add XMP metadata to a PDF, and if needed, to design and implement a new standard interface for this. In this article we will describe the current state of this task.

Contents

1 Introduction

The PDF format offers two places to store metadata. For one there is the *Info dictionary.* It is directly at the root of the PDF structure and contains key–value pairs representing document data, such as the PDF creation date (e.g., `/CreationDate (D:2022100515 3151+02’00’)`) and title (e.g., `/Title (Bearwear)`). (Recall that string constants in PDF and PostScript are enclosed in parentheses.) Various standard keys exist, such as `/Title`, `/Author` and `/CreationDate`, but the dictionary can also hold private keys. pdfTeX for example adds its banner: `/PTEX.Fullbanner (This is pdfTeX, ...)`. PDF viewers normally show a selection of the standard keys in the properties of a PDF. All TeX engines offer tools to add content to this dictionary and to change the values added automatically by the engines. In LaTeX the title and the author are normally added with the help of the hyperref package and its `pdftitle` and `pdfauthor` keys.

While the Info dictionary can hold arbitrary data, it is nevertheless only an unordered list and not well suited as a serious data container. So in PDF 1.4 a second option was added to the format: one can embed an XML file with the data and reference this file from the PDF catalog through the `/Metadata` key.[1] The format of the XML is defined as part of a framework called the *Extensible Metadata Platform* (XMP), first described in an Adobe document and now an ISO standard [3], and so commonly these metadata are referred to as *XMP metadata.*

In PDF 2.0, the XMP metadata replaces the deprecated Info dictionary. In earlier PDF versions the metadata has already been required by standards like the various PDF/A and PDF/X versions, PDF/UA (the standard for accessible PDF), and standards for electronic invoice data exchange like ZUGFeRD 2.2/ Factur-X 1.0 [1]. It is therefore quite important to have tools and interfaces to add them to a PDF. In the following we will describe various existing options and give our outlook on future plans in this area.

We assume that all source files are UTF-8 encoded and won’t mention places where 8-bit encoded files need perhaps additional care (XMP metadata in the PDF are always UTF-8 encoded).

2 Creating the XMP metadata

It is quite easy to add an XML file to a PDF and to reference it in the catalog. In all engines this can be done with a few lines of code; the small package xmpincl [9] demonstrates it for pdfLaTeX. The challenge is to correctly build the content.

- At first, as always with XML there is quite a large amount of formal syntax to understand and follow.
- PDF standards contain further demands on the content and the structure of the XML. As an example, properties that don’t count as predefined [8] must be declared in extension schemas, and if data like a title is present in the Info dictionary and in the XMP metadata they must match — something that is not easy to ensure as they use different encodings and formatting and so this requires concrete tests with PDF viewers and validators.
- Then one must decide which XMP metadata should be supported (various more or less standard name spaces exist here) and devise user interfaces for the data that can’t be detected automatically.
- User input must be sanitized, properly escaped for use in an XML file and converted to UTF-8.

In the past, two LaTeX packages took on this task. They don’t produce exactly the same XMP metadata, but the differences are small; on the whole, they settled more or less on the same set.

[1] It is possible to add more XML files and to reference them from other parts of the PDF but in this article we restrict the discussion to the document-wide data container.

doi.org/10.47397/tb/43-3/tb135fischer-xmp

Listing 1: A selection of XMP metadata added automatically by hyperxmp

```
<pdf:PDFVersion>1.5</pdf:PDFVersion>
<dc:format>application/pdf</dc:format>
<xmp:CreateDate>2022-10-06T10:27:33+02:00
  </xmp:CreateDate>
<xmp:CreatorTool>LaTeX with hyperref
  </xmp:CreatorTool>
<xmpMM:DocumentID>
  uuid:aef2b675-9b18-4d18-97f7-a3339b139000
</xmpMM:DocumentID>
```

Listing 2: hyperxmp example with additional `\hypersetup` keys

```
\documentclass{article}
\usepackage{hyperxmp}
\usepackage{hyperref}
\hypersetup
 {
  pdftitle = {Über einen die Erzeugung und
    Verwandlung des Lichtes betreffenden
    heuristischen Gesichtspunkt},
  pdfauthor={Albert Einstein},
  pdflang = {de},
  pdfmetalang={de},
  pdfdate={1905-03-17},
  pdfcontactcity={Bern},
  pdfcontactcountry={Switzerland},
  pdfissn={0003-3804},
  pdfdoi={10.1002/andp.19053220607},
 }
\begin{document}
Text
\end{document}
```

2.1 The hyperxmp package

Simply loading the hyperxmp package from Scott Pakin [7] will add XMP metadata. Listing 1 shows some selected lines.

hyperxmp supports all major compilation routes: pdfLaTeX, LuaLaTeX, XeLaTeX, LaTeX with `dvips`,[2] (u)(p)LaTeX with DVIPDFMx.

For the user interface, the package hooks into the `\hypersetup` command of hyperref. It retrieves the values of native hyperref keys like `pdftitle` and `pdfauthor` and defines new keys for a variety of additional XMP metadata. Listing 2 shows a few examples. The hyperxmp package supports quite a large set of metadata tags but has no interface to extend this set.

[2] But sadly Ghostscript has no option to add the XMP metadata as an uncompressed stream.

hyperref is used by hyperxmp not only for the user interface but also to sanitize the user input: All user input is first converted by `\pdfstringdef` to the format used in bookmarks and then back to UTF-8; it can contain arbitrary Unicode characters and all commands supported by hyperref in the bookmarks. If hyperref is missing it is loaded automatically by hyperxmp at the end of the preamble. If the document should contain XMP metadata but not links or bookmarks, load hyperref with the package option `draft`.

Individual items in the author and keyword lists should be separated by commas; if a real comma is wanted `\xmpcomma` must be used.

Some keys allow adding language variants with the command `\XMPLangAlt`:

```
\hypersetup{pdflang=de,pdftitle=Mein Titel}
\XMPLangAlt{en}{pdftitle={My title}}
```

This results in metadata using the `xml:lang` attribute:

```
<dc:title>
 <rdf:Alt>
  <rdf:li xml:lang="x-default">Mein Titel
    </rdf:li>
  <rdf:li xml:lang="de">Mein Titel</rdf:li>
  <rdf:li xml:lang="en">My title</rdf:li>
 </rdf:Alt>
</dc:title>
```

2.2 The pdfx package

The goal of the pdfx package [2], currently maintained by Ross Moore, is to support the generation of PDF/X-, PDF/A- and PDF/E-compliant documents. As XMP metadata are required by the standards they are created by the package, but it will (in part depending on the requested standard) also embed a color profile, change ToUnicode values, redefine math accents and more.

pdfx can only be used with pdfLaTeX, LuaLaTeX and XeLaTeX. (It makes use of the xmpincl package written for pdfLaTeX for the actual embedding of the XMP data and tweaks it a bit to make it compatible with the two other engines). The compilation with XeLaTeX requires the use of `--shell-escape` as pdfx calls LuaLaTeX to retrieve the creation date of the file; this can be avoided by defining `\pdfcreationdate` manually before loading pdfx, e.g.[3]

```
\def\pdfcreationdate
  {\string D:20221002224824+10'00'}
```

[3] The `\string` is needed as pdfx expects a `D` with catcode other.

Listing 3: Example input with the pdfx package

```
\begin{filecontents}[force]{\jobname.xmpdata}
 \Title{Baking through the ages}
 \Author{A. Baker\sep C. Kneader}
 \Language{en-GB}
 \Keywords{cookies\sep muffins\sep cakes}
 \Publisher{Baking International}
\end{filecontents}
\documentclass{article}
\usepackage[a-1b]{pdfx}
\begin{document}
 some text
\end{document}
```

When using pdfx the user must provide metadata in an external file with the extension `.xmpdata`. It is recommended to create this file at the beginning of the document with a `filecontents` environment. In this `.xmpdata` file the data are then given as arguments of various commands—Listing 3 shows an example. The arguments can contain arbitrary Unicode characters, and don't need to be printable in the document. Commands for non-Latin characters like `\CYRD` (cyrillic), `\hebzayin` (hebrew) or `\textarabicfa` can also be used, if enabled through package options like cyrxmp and hebxmp.

pdfx loads hyperref as it, like hyperxmp, uses `\pdfstringdef` to sanitize some of the input. It also uses hyperref to set the PDF version. As hyperref normally should be loaded late but the PDF version must be set early, getting the loading order right can get tricky.

The default setup for the XMP metadata is stored in external templates,[4] `pdfa.xmp`, `pdfx.xmp`, `pdfe.xmp`, ... By copying these templates and adjusting them it is possible to add private data. As a side effect of the processing the resulting XMP data are written to a file, `pdfa.xmpi` or `pdfx.xmpi`, which allows to feed them to an RDF validator [11].

Lists are created if the items are separated by the command `\sep`. Language support is offered through optional arguments:

```
\Title[en]{Baking through the ages}
\Keywords[de]{Kekse%
   \sep[en]Cookies%
   \sep[fr]Biscuits}
```

This will result in:

[4] Depending on the standard, pdfx will create slightly different metadata.

```
<dc:title>
 <rdf:Alt>
  <rdf:li xml:lang="en">
    Baking through the ages
  </rdf:li>
 </rdf:Alt>
</dc:title>
<dc:subject>
 <rdf:Bag>
  <rdf:li xml:lang="de">Kekse</rdf:li>
  <rdf:li xml:lang="en">Cookies</rdf:li>
  <rdf:li xml:lang="fr">Biscuits</rdf:li>
 </rdf:Bag>
</dc:subject>
```

2.3 hyperxmp and pdfx clash

The two packages can't be used in the same document. As both use low-level primitive commands to reference their XMP stream, the PDF catalog would contain two `/Metadata` entries if both packages are used, and this is invalid in a PDF:

```
/Metadata 3 0 R /Metadata 6 0 R
```

3 XMP metadata with the LaTeX PDF management support

Over the past years the LaTeX Project Team has in connection with the Tagged PDF project [6] written and released code targeting various PDF-related tasks. Some of this code is already included in the kernel through the l3pdf module of the L3 programming layer [4], the rest is provided through the external bundle pdfmanagement-testphase [5].

The goals of this code are, first, to provide abstracted, backend independent commands. So for example instead of setting the pdf version directly (depending on the engine: with `\pdfminorversion`, or `\pdfvariable minorversion` or with a `\special`), you can use `\pdf_gset_version:n{1.7}` with all supported backends. hyperref already makes use of this: when the PDF management is active the same generic driver is loaded for all backends.

The second goal of the new code (and the reason why it is called PDF *management*) is to prevent clashes like the one between hyperxmp and pdfx mentioned above, by providing interfaces for *managed* access to central PDF resources. If for example two packages try to add a `/Metadata` reference or some other resources to the catalag, then the code will ensure that the conflict is correctly resolved — either by merging the resources or by rejecting one of the requests.

The PDF management doesn't work with packages that bypass the interfaces by using primitive commands. This means neither hyperxmp nor pdfx are usable with it, and a replacement to add the important XMP metadata was needed. As an intermediate solution hyperxmp was patched but as part of task 2.3.4 of the feasibility study [6] this patch has now been replaced by proper support in the l3pdfmeta package of the pdfmanagement-testphase bundle.

3.1 Design goals for the XMP metadata support

After reviewing the existing packages the following main design goals of the new XMP metadata support have been identified:

- The dependency to hyperref should be removed. XMP metadata are not directly related to links and other interactive features and should work also in documents which don't use them.
- The standard interface should be a key–value system with `\DocumentMetadata` as the default interface command. To ease the transition from hyperxmp the `\hypersetup` keys should continue to work where possible.
- The input should support the full range of Unicode and standard commands.
- The default set of supported XMP tags should be similar to the set of the existing packages.
- There should be no need in the user input for special commands like `\xmpcomma`, `\XMLlangalt` or `\sep`. List items should be input as comma lists where "real commas" are protected by braces as usual. The language alternatives can be set with optional arguments.
- As with pdfx it should be possible to export the XMP metadata in an external document, but this should be a debug option.
- The XMP metadata should be extensible. As a proof of concept, an example document showing how to add the metadata needed for a ZUGFeRD document should be developed.

3.2 Implementation of the design goal

The new XMP metadata support has been implemented in the l3pdfmeta module and is loaded together with PDF management code. This is done by using the `\DocumentMetadata` command with key–value pairs at the beginning of the document. XMP metadata are then automatically added to the PDF they can be suppressed by setting the key `xmp` to `false` (see listing 4). This works with all engines supported by the L3 layer backend.

In accordance with the goals specified above the minimal document doesn't require hyperref: The code relies on `\text_purify:n` and other functions from the L3 layer to sanitize the input.

Listing 4: Minimal input for XMP metadata with the PDF management

```
\DocumentMetadata
  {
   %xmp=false, % no XMP
    xmp=true   % optional as default
  }
\documentclass{article}
\begin{document}
abc
\end{document}
```

XMP metadata for the PDF standard,[5] the PDF version and the language are already retrieved from keys set in `\DocumentMetadata`:

```
\DocumentMetadata
  {
   pdfstandard = a-2b,
   pdfversion  = 1.7,
   lang        = de
  }
```

At the moment other metadata still requires the use of the hyperref and the `\hypersetup` interface with the hyperxmp keys known from listing 2 — we haven't decided yet how to name and organize suitable keys in the `\DocumentMetadata` command.

As outlined in the design goals, lists are input as comma lists with real commas protected by braces as usual. Where sensible it is possible to add a language tag with an optional argument before the item. The next listing demonstrates both for the title:

```
\hypersetup
 { pdftitle=
    {[en]Baking,
     [de]{Kekse, Kuchen und Torten backen}}
```

This results in this metadata:

```
<dc:title>
  <rdf:Alt>
   <rdf:li xml:lang="en">Baking</rdf:li>
   <rdf:li xml:lang="de">Kekse,
     Kuchen und Torten backen</rdf:li>
  </rdf:Alt>
</dc:title>
```

[5] As with pdfx, declaring a standard can have further effects; for example, embed a color profile or do some validations. Be aware that LaTeX can neither ensure nor check all requirements of any given standard, and an external validator like veraPDF [10] should be used.

The XMP metadata can be exported to an external file — the default name is `\jobname.xmpi` — with a `debug` setting in `\DocumentMetadata`:

```
\DocumentMetadata
  {
   debug = { xmp-export, more debug options ... }
  }
```

Finally, first steps have been undertaken to extend the XMP metadata. To support, for example, the ZUGFeRD standard, additions to the XMP metadata are needed in three places:

- A new XML namespace with a suitable prefix must be declared.
- A new schema with declarations for the new tags must be added to the `pdfaExtension:schemas` section.
- And the data itself must be added.

For all these tasks internal functions have been defined, and a first prototype that implements the ZUGFeRD 2.2 standard exists, but it isn't yet clear what the public interface should look like.

3.3 Status and outlook

The new code supports XMP metadata at a comparable level to that provided by the existing packages. Almost all of the above design goals are already implemented. There remains some work to do to provide suitable public interfaces for certain parts. Also needed are more tests with PDF viewers and validators to check if their interpretation of the standards agrees with the code. Feedback and comments are welcome!

References

[1] Forum for Electronic Invoicing Germany. What is ZUGFeRD? `www.ferd-net.de/standards/what-is-zugferd`

[2] Hàn Thế Thành, P. Selinger, et al. *The pdfx package.* `ctan.org/pkg/pdfx`

[3] International Organization for Standardization. *ISO 16684-1:2019: Graphic technology — Extensible metadata platform (XMP) specification — Part 1: Data model, serialization and core properties.* 2nd ed., 2019. `www.iso.org/obp/ui/#!iso:std:75163:en`

[4] LaTeX Project Team. *The l3kernel package.* `ctan.org/pkg/l3kernel`

[5] LaTeX Project Team. *The pdfmanagement-testphase package.* `ctan.org/pkg/pdfmanagement-testphase`

[6] F. Mittelbach, U. Fischer, C. Rowley. LaTeX tagged PDF feasibility evaluation. `latex-project.org/publications/2020-tagged-pdf-feasibility.pdf`

[7] S. Pakin. *The hyperxmp package.* `ctan.org/pkg/hyperxmp`

[8] PDF competence center. Technote 0008: Predefined XMP properties in PDF/A-1. Technical report, 2008. `www.pdfa.org/resource/technical-note-tn0008-predefined-xmp-properties-in-pdfa-1/`

[9] M. Sneep. *The xmpincl package.* `ctan.org/pkg/xmpincl`

[10] veraPDF consortium. veraPDF—industry supported PDF/A validation. `verapdf.org`

[11] W3C. W3C RDF validation service. `www.w3.org/RDF/Validator/`

◇ Ulrike Fischer
LaTeX Project Team
Bonn
Germany
`ulrike.fischer (at) latex-project.org`

◇ Frank Mittelbach
LaTeX Project Team
Mainz
Germany
`frank.mittelbach (at) latex-project.org`

The LaTeX Tagged PDF project — A status and progress report

Frank Mittelbach, Ulrike Fischer

Abstract

The LaTeX Tagged PDF project was started in spring 2020 and announced to the TeX community by the LaTeX Team at the (online) 2020 TUG conference. This short report describes the progress and status of this multi-year project.

Contents

1 Project overview

A tagged PDF is a PDF with additional semantic structure, which improves accessibility and reuse. To enable LaTeX to create such tagged PDFs, the LaTeX Tagged PDF project was initiated in Q4 of 2019 with a feasibility study produced for Adobe [18]. This led to a commitment by Adobe to financially support the project as proposed in that study. Unfortunately, due to the COVID-19 pandemic that flared up at that time, the execution of this commitment was delayed until Q3 of 2020. Despite this delay, the LaTeX Project Team started the effort in late spring 2020 (with at that time limited resources) and the project was announced to the TeX community at the (online) TUG 2020 conference, where the team also presented the first results from Phase I of the project [19].

The LaTeX Tagged PDF project is divided into six phases, each producing immediately usable applications. This ensures both early user benefits and early feedback to the project team. The phases are roughly aligned with the bi-yearly LaTeX release cycle, with each phase being expected to take one to three LaTeX releases.

In the feasibility study, all identified project tasks are given a unique number, in order to easily cross-reference them, describe their dependencies, and arrange them in the project schedule outlined in the study. In addition to referring to tasks by name, the current report also lists these task numbers to assist readers in finding more detailed information about a particular task by looking it up in that study [18].

1.1 Phase I — Prepare the ground

The purpose of the first phase was the implementation of the core functionality inside LaTeX that forms the basis for all the later work of creating a well-tagged PDF. This phase was completed in 2021 and contained three important milestones:

- "The Hook Management System" (task 2.2.5)
- "PDF Object Support" (task 2.2.6)
- "The Automated Testing Environment" (task 2.1.2)

As part of the work on Phase I we also identified two new tasks not covered in the feasibility study:

- "Add Generic Command Hooks" (task 2.2.5(b))
- "Provide a General Configuration Point Management" (task 2.2.5(c))

1.1.1 The hook management system

The "Hook Management System" was made available to the general public in the 2020 fall release of LaTeX and enhanced LaTeX with a generic hook interface and a variety of document, shipout, file and environment hooks [11, 13, 15].

Half a year of general use of the hook management system (by the team and by many third-party developers) showed that it needed some extensions and adjustments. This resulted in the add-on task 2.2.5(b) to augment the hook management system with a generic method to automatically add hooks to third-party commands when necessary.

These generic command hooks will allow us to patch third-party code from the outside (i.e., without taking over the maintenance of abandoned but otherwise functional packages) and this way simplifying the adoption of tagged PDF. The generic command hooks were implemented and made available in the 2021 spring release of LaTeX [12].

We also identified the need for "General Configuration Point Management", similar to hook management but for configurations where only a single package or class can be in charge. This is needed to avoid packages patching into internal LaTeX commands and overwriting each other (partially) or overwriting tagging support code. For example, several packages currently attempt to alter the same internal commands of the output routine to insert some special code for footnote handling.

Conceptual work for this new task 2.2.5(c) has already been undertaken; package writer interfaces, similar to those of the hook management, will be provided during 2023.

1.1.2 PDF object support

"PDF Object Support", code to support various PDF related tasks, is in part already included in the kernel through the `l3pdf` module of the L3 programming layer [5]. Further functionality is provided through the external bundle `pdfmanagement-testphase` [16], which was released in early 2021. This will be integrated into the kernel at a later stage.

As part of this task it was necessary to work with TeX engine developers to develop some engine patches for LuaTeX and pdfTeX, in order to enable these engines to fully support PDF 2.0 Structure Destinations (XeTeX was already capable out of the box). Such structure destination provides the same view mechanism as a destination, but references a structure element instead of a page and so creates a direct connection from a link to some content. With TeX Live 2022 and current MiKTeX, structure destinations are now created automatically if the `\DocumentMetadata` command, or the `tagpdf` package, are used to create a PDF 2.0 document.

Sadly there is currently no easy way to check locally if a link points to a structure and to which one. The tag-view of Adobe Pro doesn't show them and its html export ignores the structure. To test the new feature one has to check the internal PDF structure or use an online service like ngPDF [1], a demo site for new technology to derive HTML from tagged PDF in a predictable manner, developed by the PDF Association. When using ngPDF the HTML export of a tagged PDF with structure destinations contains links to the id of a structure:

```
<h1 id="6d4ff5-1">1 abc</h1>
<a href="#6d4ff5-1">1</a>
```

Without the structure destination the link would point only to a page related target:

```
<a href="#page-0">1</a>
```

1.1.3 The automated testing environment

The "Automated Testing Environment" is integral to achieving a successful completion to the project. It is essential to build up a large test suite on which all tests can be run and verified automatically. This is because we need to modify substantially the core LaTeX code without adversely affecting the millions of existing users who expect to be able to continue to reprocess their documents without finding any unexpected visual changes. Thus, even though this code is handled directly only by the LaTeX team, its existence and stability is of utmost importance to the success of the project.

1.2 Phase II — Provide tagging of simple documents

The main goal of Phase II is to provide automatic tagging of simple documents, excluding more complicated structures such as mathematics, tables, etc. This will be achieved by setting up the necessary core code that provides general mechanisms to deal with the issues around the automatic detection of paragraph text and its correct tagging, together with enabling a subset of the standard LaTeX document elements to produce the required tags.

Work on these two essential foundations for Phase II was started earlier, in parallel to finishing Phase I:

- "Core Tagging Support" (task 2.3.1); and
- Support for "Automated Paragraph Tagging" (task 2.3.2).

The main objective of these tasks is to provide the necessary infrastructure for the automatic tagging of relatively non-complex documents. The current focus is thus on the remaining major task for Phase II:

- "Implement tagging for the basic document elements of LaTeX" (task 2.3.3).

Work on Phase II took slightly longer than initially estimated due to additionally identified tasks that are either prerequisites for a successful completion of Phase II, or necessary for later phases and, for one reason or another, were best undertaken now, in parallel. We expect to close out Phase II be the end of 2022 with an out-of-sequence release of the `latex-lab` bundle, which was added in June 2022 to enable safe experimentation with new project code without disrupting workflows using production LaTeX [9].

1.2.1 Core tagging support

Tagging a PDF requires writing and managing various objects and literals in the PDF. The needed "Core Tagging Support" code is currently available as an add-on package (the package `tagpdf` [2]), since this allows for safe experimentation by those who wish to have tagged PDF output now, but without disrupting any user workflows. Once it is thoroughly tested, the code will be integrated into the kernel (this will form its own task in a later phase). How to use the code to create simple tagged PDF's was described by Ulrike Fischer at the (online) 2021 TeX Users Group conference [3].

1.2.2 Automated paragraph tagging

Large parts of standard documents consist of simple paragraphs. For the success of the project it is of utmost importance that such paragraphs are tagged automatically and that paragraphs split over pages are handled correctly. The kernel extensions needed for such "Automated Paragraph Tagging" were finished in time for the 2021 spring release of LaTeX and announced at the (online) 2021 TUG conference by Frank Mittelbach [17]. They use marks and new hooks at the beginning and end of paragraphs [14].

1.2.3 Tagging for basic document elements

The goal of this task is to make standard LaTeX document elements tagging-aware, so that all the structural information they encapsulate is automatically transferred into an appropriate tag structure (including attributes) in the resulting PDF document. In the project schedule this task is split between phases II and III, starting in Phase II by concentrating on the high-level structural elements such as links, headers and footers, headings, lists, footnotes and tables of contents. The `tagpdf` package and the PDF management code already implement the automatic tagging of hyperlinks and the tagging of headers and footers (as artifacts). Automatic footnote tagging (with links) including special cases, such as footnotes broken across columns or pages, tagging of lists and tagging of tables of contents will be deployed with the release of `latex-lab` at the end of 2022. Later in Phase III (in 2023) the remaining basic document structures will be added, leaving more complex structures, such as tables, to later phases.

In this context, "automatic tagging" means that "identified document elements will be mapped using a default mapping to PDF tags without manual adjustments or fine-grained flexibility." Such flexibility will of course eventually be necessary in order to produce the highest quality of tagged PDF; therefore, this flexibility had to be catered for already in the underlying support code, which led to a new task:

- "Design and implement a general key/value interface" (task 2.3.3b).

This was identified as an additional prerequisite for successfully completing tasks 2.3.3 and 2.3.5. It must be made possible to extend the optional argument of standard commands and environments, for example, for sectioning and captions to accept key/value arguments to specify alternative text.

The task also includes the design and implementation of a general template mechanism for commands and environments using the key/value method for configuration. However, exposing these concepts on the user and package developer levels will require the design of interfaces for their configuration and manual overwriting, both of which are parts of later phases.

1.3 Preparatory work for tasks in Phases III and IV

For a number of technical and practical reasons we have diverged from the original schedule layout and have already started work on tasks planned for later phases. These include:

- "Design and implement an extended cross-reference mechanism for LaTeX" (task 2.2.2)
- "Provide an interface for specifying all types of document metadata" (task 2.3.4)
- "Design and implement hyperlinking and move it into the LaTeX kernel" (task 2.2.3)
- "Standards compliance" (task 2.3.9)

1.3.1 Interface to document metadata

The status and outlook on the implementation of document metadata (task 2.3.4) is described in a separate article in the current issue [4].

1.3.2 Hyperlinking improvements

User interfaces (and backend code) for hyperlinking facilities are currently provided mainly by the `hyperref` package. With the new PDF Object Support described above, large parts of the backend code have already been moved into the LaTeX kernel or into the `pdfmanagement-testphase` package. At the time when structures such as footnotes, headings, and tables of contents are made tagging-aware, native hyperlinking support will be added as well, and the no-longer-needed patches in `hyperref` suppressed.

1.3.3 Standards compliance

Support for various PDF/A standards is provided by the `pdfmanagement-testphase` package (`l3pdfmeta` module). The code will add the typically-needed color profiles and PDF objects, and suppress forbidden actions such as JavaScript code.

It should be noted that LaTeX cannot check all requirements of a standard and that an external validator such as veraPDF should be used for this. Support for PDF/X standards is currently only provided in the form of XMP metadata entries.

1.4 Cross-phase tasks

A number of tasks require attention and action across all phases. Up to now these are:

- "Define a change strategy to safely extend LaTeX without causing serious issues for the worldwide user base" (task 2.1.1)
- "Developer acceptance testing for finished tasks" (task 2.4.1)
- "Coordinate updates to external packages" (task 2.4.3)

Altering and enhancing LaTeX without disrupting existing documents and workflows is an important goal of the project. For this a number of tools have been implemented:

- The `\DocumentMetadata` interface allows tagged documents and non-tagged documents to be processed by the same LaTeX format by changing only one line.
- Experimental code is kept first in external packages such as `pdfmanagement-testphase` and `tagpdf`, or the `latex-lab` bundle.
- The `firstaid` package allows us to temporarily patch external packages if it turns out that they are incompatible with a change.
- The `latex-dev` releases give package authors time and opportunity to test changes and report problems, and the LaTeX team time to contact package authors and coordinate updates.
- The package `tagpdf-base` provides dummy versions of the core tagging commands, thus supporting the writing of commands and environments which are properly tagged if the user activates tagging, but which also work without tagging.
- The `\IfDocumentMetadataTF` kernel command allows testing if the new interface has been used in a document.
- The `\MakeLinkTarget` kernel command provides a dummy version of the command used by `hyperref` that creates anchors, and so allows writing commands and environments with built-in hyperlinking features which are activated if the user loads `hyperref`.

1.5 Interface to project code for users

As part of the metadata task (2.3.4), we provided a `\DocumentMetadata` command in the LaTeX kernel to be used as the very first declaration in a document (i.e., before `\documentclass`). This allows us to load the PDF management code and enable tagging and other project-related code. In short, by using this declaration the user indicates that this is a document to which tagging should be applied. That is, it serves a similar role as the switch from `\documentstyle` (old LaTeX 2.09 pre-1994) to `\documentclass` (modern LaTeX). This eases the transition and allows old and new code to coexist.

In this way, we also avoid users having to load special packages to test new features, instead everything boils down to giving a simple line

```
\DocumentMetadata{testphase=phase-II,...}
```

at the start of the document which then loads everything necessary to make use of currently-available results from a particular phase.

This interface has been deployed in LaTeX 2022 June release and from that point onwards it is available to every LaTeX user.

1.6 Summary of current status

Phase I of the project was completed previously and the results are deployed and in general use today.

Phase II is near completion, with most tasks implemented and deployed, and an expected closeout with the release of the `latex-lab` bundle at the end of 2022. This will then enable automatic tagging of LaTeX documents with a (still fairly) simple element structure by adding the aforementioned

```
\DocumentMetadata{testphase=phase-II,...}
```

at the top of a document.[1] Thus the major aim of Phase II is to enable many existing LaTeX documents to be reprocessed to produce tagged, and hence accessible, PDF output with no change to the source file other than adding a line like the above.

The goal of the later phases is then to expand this scope with more and more document elements being recognized, and to support adjustments to the tagging, thereby increasing the quality of the tagged PDF output. Eventually, the temporary interface within `\DocumentMetadata`, responsible for loading the `tagpdf` support package, will also no longer be necessary.

Overall, we can confidently state that the project progress is in good shape and within the main plan boundaries and will continue to be so.

2 Software releases

In the period between 2020–Q4 and 2022–Q4 the team has implemented and distributed five main LaTeX releases that have a direct bearing on the progress of the project. Important features of the releases are summarized below; the releases also contain other improvements not directly related to the Tagged PDF project.

LaTeX Release 2020-10, see [6]

- Provide the new hook management for LaTeX (task 2.2.5)
- Move the `xparse` interfaces to the kernel (needed for various later tasks to provide document-level interfaces)

[1] The data to place into the `\DocumentMetadata` argument is temporary at this point and will change over the course of the project, e.g., `testphase=phase-II` means apply the code for Phase II — something that will not be necessary once the code is finalized and integrated with the LaTeX kernel.

LaTeX Release 2021-06, see [7]

- Extending hook management to paragraphs (needed for task 2.3.2)
- Extending hook management to commands (needed for Phases II & III tasks)

LaTeX Release 2021-11, see [8]

- Consolidation release
- Corrections and improvements to the hook management system after extensive use by the project team and by third-party developers

LaTeX Release 2022-06, see [9]

- `\DocumentMetadata` interface (needed for task 2.3.4)
- Introduction of the `latex-lab` bundle (needed to allow safe user experimentation with new functionality from the project)
- First part of the new key/value handling in the kernel (needed for several tasks)

LaTeX Release 2022-11, see [10]

- Auto-detecting new key/value arguments, e.g., in `\section` or `\caption` (implements task 2.3.3b; needed for several other tasks)

References

[1] DualLab. Next generation PDF. `ngpdf.com`

[2] U. Fischer. *The tagpdf package.* `ctan.org/pkg/tagpdf`

[3] U. Fischer. On the road to Tagged PDF: About StructElem, marked content, PDF/A and squeezed Bärs. *TUGboat* 42(2):170–173, 2021. `https://doi.org/10.47397/tb/42-2/tb131fischer-tagpdf`

[4] U. Fischer, F. Mittelbach. Adding XMP metadata in LaTeX. *TUGboat* 43(3):263–267, 2022. `https://doi.org/10.47397/tb/43-3/tb135fischer-xmp`

[5] LaTeX Project Team. *The L3kernel package.* `ctan.org/pkg/l3kernel`

[6] LaTeX Project Team. LaTeX news, issue 32, 2020. `www.latex-project.org/news/latex2e-news/ltnews32.pdf`

[7] LaTeX Project Team. LaTeX news, issue 33, 2021. `www.latex-project.org/news/latex2e-news/ltnews33.pdf`

[8] LaTeX Project Team. LaTeX news, issue 34, 2021. `www.latex-project.org/news/latex2e-news/ltnews34.pdf`

[9] LaTeX Project Team. LaTeX news, issue 35, 2022. `www.latex-project.org/news/latex2e-news/ltnews35.pdf`

[10] LaTeX Project Team. LaTeX news, issue 36 — draft, 2022. `www.latex-project.org/news/latex2e-news/ltnews36.pdf`

[11] LaTeX Project Team. *LaTeX's hook management*, 2022. `mirrors.ctan.org/macros/latex/base/lthooks-doc.pdf`

[12] LaTeX Project Team. *The ltcmdhooks module*, 2022. `mirrors.ctan.org/macros/latex/base/ltcmdhooks-doc.pdf`

[13] LaTeX Project Team. *The ltfilehook documentation*, 2022. `mirrors.ctan.org/macros/latex/base/ltfilehook-doc.pdf`

[14] LaTeX Project Team. *The ltpara.dtx code*, 2022. `mirrors.ctan.org/macros/latex/base/ltpara-doc.pdf`

[15] LaTeX Project Team. *The ltshipout package*, 2022. `mirrors.ctan.org/macros/latex/base/ltshipout-doc.pdf`

[16] LaTeX Project Team. *The pdfmanagement-testphase package*, 2022. `ctan.org/pkg/pdfmanagement-testphase`

[17] F. Mittelbach. Taming the beast — Advances in paragraph tagging with pdfTeX and XeTeX (2021): Automatic paragraph tagging with the pdfTeX and XeTeX engine now possible. `www.latex-project.org/news/2022/09/06/TUG-online-talks-21-22/`

[18] F. Mittelbach, U. Fischer, C. Rowley. LaTeX Tagged PDF feasibility evaluation. `latex-project.org/publications/2020-tagged-pdf-feasibility.pdf`

[19] F. Mittelbach, C. Rowley. LaTeX Tagged PDF—a blueprint for a large project. *TUGboat* 41(3):292–298, Nov. 2020. `latex-project.org/publications/2020-FMi-TUB-tb129mitt-tagpdf.pdf`

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

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

LaTeX News

Issue 36, November 2022

Contents

Introduction

The 2022-11 release of LaTeX is largely a consolidation release where we made a number of minor improvements to fix some bugs or improve one or the other interface.

The only really important functionality that was added is described in the next section: the ability to easily define document-level commands and environments that accept a key/value list in one of its (usually optional) arguments, including the ability to determine if the argument does in fact contain such a key/value list or just a single “classical” value.

For the “Tagged LaTeX Project” this functionality is very important because many document-level commands will need to accept such key/value lists, for example, to specify alternative text or overwrite default tagging if that becomes necessary in a document.

Auto-detecting key/value arguments

To allow extension of the core LaTeX syntax, ltcmd now supports a `=...` modifier when grabbing arguments. This modifier instructs LaTeX that the argument should be passed to the underlying code as a set of key/values. If the argument does not “look like” a set of key/values, it will be converted into a single key/value pair, with the argument to `=` specifying the name of that key. For example, the `\caption` command could be defined as

```
\DeclareDocumentCommand\caption
      {s ={short-text}+O{#3} +m}
      {...}
```

which would mean that if the optional argument does *not* contain key/value data, it will be converted to a single key/value pair with the key name `short-text`.

Arguments which begin with `=,` are always interpreted as key/values even if they do not contain further `=` signs. Any `=` signs enclosed within `$...$` or `\(...\)`, i.e. in inline math mode, are ignored, meaning that only `=` outside of math mode will generally cause interpretation as key/value material.

In case the argument contains a “textual” `=` sign that is mistaken as a key/value indicator you can hide it using a brace group as you would do in other places, e.g.,

```
\caption[{Use of = signs}]
        {Use of = signs in optional arguments}
```

However, because `=` signs in math mode are already ignored, this should seldom be necessary.

A note for font package developers

Encoding subsets for TS1 encoded fonts

The text companion encoding `TS1` is unfortunately not very faithfully supported in fonts that are not close cousins to the Computer Modern fonts. It was therefore necessary to provide the notion of “sub-encodings” on a per font basis. These sub-encodings are declared for a font family with the help of a `\DeclareEncodingSubset` declaration, see [5] for details.

Maintainers of font bundles that include `TS1` encoded font files should add an appropriate declaration into the corresponding `ts1`*family*`.fd` file, because otherwise the default subencoding is assumed, which is probably disabling too many glyphs that are actually available in the font.[1] *(github issue 905)*

[1]The LaTeX format contains declarations for many font families already. This was done in 2020 to quickstart the use of the

doi.org/10.47397/tb/43-3/tb135ltnews36

New or improved commands

Better language handling for case-changing commands

The commands `\MakeUppercase`, `\MakeLowercase` and `\MakeTitlecase` now automatically detect the locale currently in use when babel is loaded. This allows automatic adjustment of letter mappings where appropriate. They also accept a leading optional argument. This accepts a key–value list of control settings. At present, there is one key available: `locale`, which can also be accessed via the alias `lang`. This is intended to allow local setting of the language, which can be done using a BCP-47 descriptor. For example, this could be used to force Turkish case changing in otherwise English input

```
\MakeUppercase[lang = tr]{Ragıp Hulûsi Özdem}
```

yields RAGIP HULÛSİ ÖZDEM.

Code improvements

Support for slanted small caps in the EC fonts

For some time LaTeX has supported the combination of the shapes small caps and italic/slanted. The EC fonts contain slanted small caps fonts but using them required the loading of an external package. Suitable font definitions have now been added to t1cmd.fd and so from now on

```
\usepackage[T1]{fontenc}
...
 \textsc{\textsl{Slanted Small Caps}};
 \textsc{\textit{Italic Small Caps}};
 \bfseries
 \textsc{\textsl{Bold Slanted Small Caps}};
 \textsc{\textit{Bold Italic Small Caps}}.
```

will give the expected result: *SLANTED SMALL CAPS*; *ITALIC SMALL CAPS*; ***BOLD SLANTED SMALL CAPS***; ***BOLD ITALIC SMALL CAPS***.

Given that the Computer Modern fonts in `T1` do not have real italic small caps but only slanted small caps, the latter is substituted for the former. This is why both work in the above, but there is no difference between the two (and you get a substitution warning for the `\textit\textsc` shape combination). *(github issue 782)*

EC sans serif at small sizes

The EC (T1 encoded Computer Modern) sans serif fonts have errors at small sizes: the medium weight is bolder and wider than the bold extended. This makes them unusable at these small sizes. The default `.fd` file has therefore been adjusted to use a scaled down 8pt font instead. *(github issue 879)*

Improve font series handling with incorrect `.fd` files

By convention, the font series value is supposed to contain no `m`, unless you refer to the "medium" series (which is represented by a single `m`). For example, one should write `c` for "medium weight, condensed width" and not `mc`. This was one of the many space-conserving methods necessary in the early days of LaTeX 2ε.

Some older `.fd` files do not obey that convention but use `mc`, `bm`, etc., in their declarations. As a result, some font selection scheme functionality was not working when confronted with such `.fd` files. We have therefore augmented `\DeclareSymbolFont` and `\SetSymbolFont` to strip any surplus `m` from their series argument so that they do not unnecessarily trigger font substitutions. Regardless of this support such `.fd` files should get fixed by their maintainers. *(github issue 918)*

Detect nested `minipage` environments

Nesting of `minipage` environments is only partially supported in LaTeX and can lead to incorrect output, such as overfull boxes or footnotes appearing in the wrong place; see [1, p. 106]. However, until now there was no warning if that happened. This has been changed and the environment now warns if you nest it in another `minipage` environment that already contains footnotes. *(github issue 168)*

Robust commands in package options

With the standard key-based option handler added in the last release, or with contributed packages offering similar features, users may expect to be able to use a package option such as `[font=\bfseries]`. Previously this failed with internal errors as the option list was expanded via `\edef`. This has now been changed to use the existing command `\protected@edef` so that any LaTeX robust command should be safe to pass to a key value option. *(github issue 932)*

Improve l3docstrip integration into docstrip

In 2020 we merged l3docstrip.tex into docstrip.tex to support the `%<@@=`⟨*module*⟩`>` syntax of expl3; see [2]. However, this support was incomplete, because it didn't cover docstrip lines of the form `%<+...>` or `%<-...>`. This was never noticed until now, because usually `%<*...>` blocks are used. Now all lines in a `.dtx` file are subject to the `@@` replacement approach. *(github issue 903)*

LuaTeX callback efficiency improvement

The mechanism for providing the `pre/post_mlist_to_hlist_filter` callbacks in LuaTeX has been improved to make it more reusable and to avoid overhead if these callbacks are not used. *(github issue 830)*

Rule-based ordering for LuaTeX callback handlers

In LuaLaTeX the callback handlers used to be called in the order in which they were registered in, but this was often rather fragile. It depends a lot on the load order and any attempts to enforce a different order required unregistering and reregistering the handlers to be reordered. Additionally, even if some ordering constraints where enforced that way, another package loaded later could accidentally overwrite it.

symbols in the kernel, but it is really the wrong place for such declarations. Thus, for new fonts the declarations should be placed into the corresponding `.fd` files.

To improve this, we now order the callback handlers based on ordering rules similar to the hook rules.

When registering a callback which should run before or after another callback, `luatexbase.declare_callback_rule` can now be used to record this ordering constraint. For example

```
luatexbase.add_to_callback
 ('pre_shaping_filter', my_handler, 'my_name')
luatexbase.declare_callback_rule
 ('pre_shaping_filter',
           'my_name', 'before', 'other_name')
```

will ensure that `my_handler` will always be called before the handler registered as `other_name`.

This also means that the order in which callbacks are registered no longer implicitly defines an order. Code which relied on this implicit order should now define the order rules explicitly.

Bug fixes

Prevent TEX from losing a `\smash`

When TEX is typesetting a fraction, it will rebox the material in either the numerator or denominator, depending on which is wider. If the repackaged part consists of a single box, that box gets new dimensions and if it was built using a `\smash` that effect vanishes (because a smash is nothing other than zeroing some box dimension, which now got undone). For example, in the line

```
\frac{1}{2} = \frac{1}{\smash{2^X}}
           \neq \frac{100}{\smash{2^X}}
```

the 2 in the denominators was not always at the same vertical position, because the second `\smash` was ignored due to reboxing:

$$\frac{1}{2} = \frac{1}{2^X} \neq \frac{100}{2^X}$$

The differences are subtle but noticeable. This is now corrected and the `\smash` is always honored. Thus now you get this output:

$$\frac{1}{2} = \frac{1}{2^X} \neq \frac{100}{2^X}$$

(github issue 517)

Resolve an issue with `\mathchoice` and `localalphabets`

The code for keeping a number of math alphabets local (introduced in 2021; see [3]) used `\aftergroup` to do some cleanup actions after a formula had finished. Unfortunately, `\aftergroup` can't be used inside the arguments of the `\mathchoice` primitive and as a result one got low-level errors if the freezing happened in such a place. The implementation was therefore revised to avoid the `\aftergroup` approach altogether. *(github issue 921)*

Reporting of unused global options when using key/value processing

Using the new key/value option processor did not properly report any unused global options when it was used in handling class options. This has now been corrected. *(github issue 938)*

Changes to packages in the graphics category

Fix a `\mathcolor` bug

The `\mathcolor` command introduced in [4] needs to scan for following sub- and superscripts, but if it did so at the end of an alignment cell, e.g., in a `array` environment, the `&` was evaluated too early, causing some internal errors. This is now properly guarded for. *(github issue 901)*

Changes to packages in the tools category

array: Correctly identify single-line m-cells

Cells in m-columns that contain only a single line are supposed to behave like single-line p-cells and align at the same baseline. To test for the condition, array used to compare the height of the cell to the height of the strut used for the table rows. However, the height of that strut depends on the setting of `\arraystretch` and if you made this negative (or very large) the test came out wrong. Therefore, we now test against the height of a normal strut to ensure that single-line cells are correctly identified as such (unless their content is truly very tall, in which case aligning is pointless anyway). *(github issue 766)*

References

[1] Leslie Lamport. LATEX: A Document Preparation System: User's Guide and Reference Manual. Addison-Wesley, Reading, MA, USA, 2nd edition, 1994. ISBN 0-201-52983-1. Reprinted with corrections in 1996.

[2] LATEX Project Team: *LATEX 2ε news 32*. `https://latex-project.org/news/latex2e-news/ltnews32.pdf`

[3] LATEX Project Team: *LATEX 2ε news 34*. `https://latex-project.org/news/latex2e-news/ltnews34.pdf`

[4] LATEX Project Team: *LATEX 2ε news 35*. `https://latex-project.org/news/latex2e-news/ltnews35.pdf`

[5] LATEX Project Team: *LATEX 2ε font selection*. `https://latex-project.org/help/documentation/`

Markdown 2.17.1: What's new, what's next?

Vít Novotný

Abstract

In this article, we introduce new features developed for the Markdown package and ideas for its future.

The article is divided into four sections. In the first three sections, we introduce the new features to three different audiences of the Markdown package:

1. the writers, who type content in Markdown;
2. the coders, who prepare templates and solutions;
3. the developers, who make the package better.

In Section 4, we outline the roadmap for the next major version of the Markdown package.

1 Writer's newsletter

In this section, we introduce four new Markdown tags, which you can use to format your manuscripts.

1.1 Superscripts and subscripts

Use superscripts and subscripts to write ordinal indicators, exponents, or atomic valencies. Since version 2.16.0, the Markdown package has supported the `superscripts` and `subscripts` options:[1]

```
\documentclass{article}
\usepackage[superscripts, subscripts]
           {markdown}
\begin{document}
\begin{markdown}
2^10^ is 1024.
H~2~O is a liquid.
\end{markdown}
\end{document}
```

Output:

2^{10} is 1024.
H_2O is a liquid.

1.2 Strike-throughs

Use strike-throughs to denote information that is no longer accurate. Since version 2.16.0, the Markdown package has supported the `strikeThrough` option:[2]

```
\documentclass{article}
\usepackage[strikeThrough]{markdown}
\begin{document}
\begin{markdown}
Under his pillow P'raps found
~~A cake that weighed a half a pound.~~
A plenty of space to roll around.
\end{markdown}
\end{document}
```

Output:

Under his pillow P'raps found
~~A cake that weighed a half a pound.~~
A plenty of space to roll around.

[1] See `https://github.com/witiko/markdown/pull/162`
[2] See `https://github.com/witiko/markdown/pull/160`

1.3 Fancy lists

In lists, it can be important to display item labels exactly as you wrote them. Since version 2.16.0, the Markdown has supported the `fancyLists` option:[3]

```
\documentclass{article}
\usepackage[fancyLists]{markdown}
\begin{document}
\begin{markdown}
You are:
a) awesome
b) brilliant
c) charming
\end{markdown}
\end{document}
```

Output:

You are:
a) awesome
b) brilliant
c) charming

2 Coder's newsletter

In this section, we introduce a new API for reacting to YAML metadata and user-defined syntax extensions.

2.1 Building better APIs with YAML

In our previous article, [3, Section 2.1] we showed how we can react to YAML metadata in Markdown documents. However, our approach used a low-level API that required use of the expl3 programming language. Since Markdown 2.16.0, the `\markdownSetup` LaTeX command has supported the `jekyllDataRenderers` key, which provides a high-level API for reacting to YAML metadata without the need to use expl3:[4]

```
\documentclass{article}
\usepackage[jekyllData]{markdown}
\newtoks\abstract \newtoks\authors
\markdownSetup {
    jekyllDataRenderers = {
        abstract = {\abstract={#1}},
        title = {\global\title{#1}},
        /authors/* = {%
            \authors=\expandafter{%
                \the\authors \and #1}%
        }, year = {%
            \global\date{%
                One year after
                \the\numexpr(#1-1)\relax}%
        },
    }, renderers = {
        jekyllDataEnd = {
            \global\author{\the\authors}%
            \maketitle \section*{Abstract}
                       \the\abstract
        },
    },
}
```

[3] See `https://github.com/witiko/markdown/pull/168`
[4] See `https://github.com/witiko/markdown/pull/175`

```
\begin{document}
\begin{markdown*}{expectJekyllData}
title: 'This is a title: with a colon'
authors: [Jane Doe, John Doe]
year: 2022
abstract: |
 This is the
 abstract

 It contains
 two paragraphs.
\end{markdown*}
\end{document}
```

Output:

This is a title: with a colon

Jane Doe John Doe

One year after 2021

Abstract

This is the abstract
It contains two paragraphs.

2.2 User-defined syntax extensions

Since version 2.17.0, the Markdown package has supported user-defined syntax extensions, which you can use to customize Markdown to your tastes:[5]

```
\documentclass{article}
\usepackage{soul}
\begin{filecontents}
   [nosearch, noheader, overwrite]
   {strike-through.lua}
local strike_through = {
   api_version = 2,
   grammar_version = 1,
   finalize_grammar = function(reader)
      local nonspace, doubleslash
      nonspace = lpeg.P(1) - lpeg.S("\t ")
      doubleslash = lpeg.P("//")

      local function between(p, sep)
         ender = lpeg.B(nonspace) * sep
         return (sep * #nonspace
               * lpeg.Ct(p * (p - sep)^0)
               * sep)
      end

      local read_strike_through = between(
         lpeg.V("Inline"), doubleslash
      ) / function(s)
         return {"\\st{", s, "}"}
      end

      reader.insert_pattern(
         "Inline after Emph",
         read_strike_through)
      reader.add_special_character("/")
   end
}
return strike_through
\end{filecontents}
\usepackage{markdown}
\begin{document}
\begin{markdown*}
      {extension = strike-through.lua}
Under his pillow P'raps found
//A cake that weighed a half a pound.//
A plenty of space to roll around.
\end{markdown*}
\end{document}
```

Output same as in Section 1.2

For more information about syntax extensions, see the technical documentation of the Markdown package [2, Section 2.1.2] and the article about parsing complex data formats in Lua by Henri Menke. [1]

3 Developer's newsletter

In this section, we introduce new reflection capabilities and discuss a recent code clean-up.

3.1 Reflection of options and renderers

In versions 2.15.0 and 2.15.3, the Markdown package has received reflection capabilities that allowed it to take a look in a mirror and inspect itself.[6]

Using reflection, we have automated parts of the code that were previously hand-written. These include parts responsible for type-checking options, passing options from plain TEX to Lua, and defining high-level interfaces for LATEX and ConTEXt.

3.2 Refactoring TEX and Lua code

In patch versions 2.15.1 through 2.15.4, we focused on cleaning up the code of the Markdown package. In the following, we discuss the major changes.

In version 2.15.3, we separated a part of the Markdown package into its own separate package

[5] See https://github.com/witiko/markdown/pull/182

[6] See https://github.com/witiko/markdown/pull/137

called lt3luabridge.[7] With lt3luabridge, you can execute Lua code in TeX engines other than LuaTeX.

Also in version 2.15.3, we separated built-in syntax extensions such as subscripts, superscripts, and strike-throughs from the base grammar of markdown.[8] This change cut the development time of new syntax extensions in half and paved the way for the introduction of user-defined syntax extensions in Markdown 2.17.0 (see Section 2.2).

In version 2.15.4, we replaced all calls to the xstring and keyval packages with built-in functions from the expl3 programming language.[9]

4 Roadmap for Markdown 3.0.0

The next major version of Markdown will be 3.0.0. Markdown 3.0.0 will remove features that have been deprecated in Markdown 2.X.Y, such as the on-disk caching of conversion outputs and the leftover interfaces for what is now the lt3luabridge package (see Section 3.2). Furthermore, Markdown 3.0.0 should also make the base grammar of markdown compliant with the CommonMark standard and freeze it, so that authors of user-defined syntax extensions (see Section 2.2) do not have to aim at a moving target.

Before Markdown 3.0.0, all syntax extensions that have been implemented to the upstream lunamark library should be ported to the Markdown package as well.[10] Furthermore, all improvements to the high-level interface for LaTeX that we have discussed in our previous article [3, sections 3.3 and 3.4] should also be implemented.[11] Finally, the user manual of Markdown should be typeset using the Markdown package and TeX4ht rather than Pandoc, which will allow us to automatically generate parts of the user manual using reflection (see Section 3.1).[12]

[7] See `https://ctan.org/pkg/lt3luabridge`
[8] See `https://github.com/witiko/markdown/pull/143`
[9] See `https://github.com/witiko/markdown/issues/96`
[10] See `https://github.com/witiko/markdown/issues/123`, `https://github.com/witiko/markdown/issues/126` and `https://github.com/witiko/markdown/issues/173`
[11] See `https://github.com/witiko/markdown/issues/107` and `https://github.com/witiko/markdown/issues/121`
[12] See `https://github.com/witiko/markdown/issues/135` and `https://github.com/witiko/markdown/issues/184`

References

[1] H. Menke. Parsing complex data formats in LuaTeX with LPEG. *TUGboat* 40(2):129–135, 2019. `tug.org/TUGboat/tb40-2/tb125menke-lpeg.pdf`

[2] V. Novotný. A Markdown interpreter for TeX. Version 2.17.1-35-g2848cb5 (2022-10-15). `mirrors.ctan.org/macros/generic/markdown/markdown.pdf`

[3] V. Novotný, D. Rehák, et al. Markdown 2.15.0: What's new? *TUGboat* 43(1):10–15, 2022. `tug.org/TUGboat/tb43-1/tb133novotny-markdown.pdf`

◇ Vít Novotný
Studená 453/15
Brno, 638 00
Czech Republic
`witiko (at) mail dot muni dot cz`
`github.com/witiko`

Mapping to individual characters in expl3

Joseph Wright

It is natural to think that separating a word up into individual characters is an easy operation. It turns out that for the computer this isn't really the case. If we look at a system that natively understands Unicode (like XƎTeX or LuaTeX), most of the time one 'character' is stored as one codepoint. A codepoint is a single character entity for a Unicode programme. For example, if we take the input 'café' in a file saved as UTF-8, it is made up of four codepoints:

```
U+0063 (LATIN SMALL LETTER C)
U+0061 (LATIN SMALL LETTER A)
U+0066 (LATIN SMALL LETTER F)
U+00E9 (LATIN SMALL LETTER E WITH ACUTE)
```

So we could, in XƎTeX/LuaTeX, use a simple mapping to grab one character at a time from this word and do stuff with it. However, that's not always the case. Take for example 'Spın̈al Tap': the dotless-i is a single codepoint, but there is no codepoint for an umlauted-n. Instead, that is represented by two codepoints: a normal n and a combining umlaut. As a user, it's clear that we'd want to get a single 'character' here. So there's clearly more work to do.

Luckily, this is not just a TeX problem and the Unicode Consortium have thought about it for us. They provide a data file and rules that describe how to divide input into *graphemes*: 'user perceived characters'. So 'all' that is needed is to examine the input using these rules, and to divide it up so that 'characters' stay together.

For pdfTeX, there's an additional wrinkle: it uses bytes, not codepoints, and so if we use a naïve TeX mapping, we would divide up any codepoint outside the ASCII range into separate bytes: not good. Luckily, the nature of codepoints is predictable: all that is needed is to examine the first byte and collect the right number of further bytes to re-combine into a valid codepoint.

This work isn't something the average end user wants to do. Luckily, they don't have to as the LaTeX team have worked on this and created a suitable set of expl3 functions to do it: `\text_map_function:nN` and `\text_map_inline:nn`.

For example, we can do (mapping each character to printing itself in parentheses):

```
\ExplSyntaxOn
\text_map_inline:nn
  { Spın̈al ~ Tap } { (#1) }
\ExplSyntaxOff
```

and get

(S)(p)(ı)(n̈)(a)(l)( )(T)(a)(p)

in any TeX engine — assuming we are set up to *print* the characters, of course. Getting the right fonts is an independent issue from parsing the input.

Taking a more 'serious' example (and one that is going to use LuaTeX for font reasons), we might want to map over Bangla text: I'm going to use ন্দ্রকিন্দ্র as my example. Our `\text_map_inline:nn` function divides up the characters correctly: (ন্)(দ্)(র)(কি)(ন্)(দ্)(র).

In contrast, the generic `expl3` token-list function `\tl_map_inline:nn` gives: (ন)(◌্)(দ)(◌্)(র)(ক)(◌ি)(ন)(◌্)(দ)(◌্)(র), which is a very odd result. In short: Unicode characters are neither bytes nor tokens.

(If you want to try that demo yourself, you'll need a document preamble that can work properly with Bangla: I'm using

```
\usepackage{fontspec}
\newfontface\harfbangla
  {NotoSansBengali-VariableFont_wdth,wght.ttf}
  [Renderer = HarfBuzz, Script = Bengali]
```

then using `\harfbangla` in a brace pair around my demonstrations. Finding a monospaced font that properly renders Bangla is … tricky.)

So, as you can see, mapping to 'real' text is easy with expl3: you just need to know that the tools are there.

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

doi.org/10.47397/tb/43-3/tb135wright-grapheme

Typesetting external program code and its output: hvextern

Herbert Voß

Abstract

When writing a book with a mathematical, scientific or technical background, the output of programs is often inserted as text or an illustration; in many cases, also with complete or partial indication of the respective source code. As an author, you have the problem of keeping such external sample programs in sync with the current manuscript. If you keep the source code in the book manuscript itself, and create the external examples at the same time as typesetting the main document, you can be sure the code and output stay consistent.

1 Introduction

If you use LaTeX to write a book about LaTeX, you can easily insert the output of the examples directly in the main document. [2] This does not necessarily mean that the examples will also work as small individual documents. All examples in a larger book use the main document's preamble, which is not available to a reader of the book.

It usually makes more sense to create examples as separate documents or programs that are independent of the main document. To do this, the complete source code is written from the main document into an external file, which is then processed using a specified program and the result is integrated back into the main document as a PDF, PNG, text, or whatever form is appropriate. From the entire source code of the example, you can use markers to place the essential lines of code in the main document before or next to the example output.

The output of the following examples was generated "on-the-fly" when compiling this *TUGboat* article. Any change in the example code therefore automatically led to updated output during the next compilation process.

To begin, first we'll show a short example: *TUGboat* is normally compiled with pdfLaTeX, so there are problems with an example that absolutely requires the use of XeLaTeX. The example must be created externally and the output integrated as a PDF. It makes more sense to do this from the main document and include the output directly, as shown here: 美好的一天.

First published in *Die TeXnische Komödie* 2/2022, pp. 30–60. Translation by the author.

This is made possible by the hvextern package, which defines only one environment and one command. [5]

The corresponding code for the above inline example is:

Inline example
```
[...] as shown here:
\begin{externalDocument}[
  compiler=xelatex, inline, runs=2, force,
  grfOptions={height=8pt}, crop, cropmargin=0,
  cleanup, docType=latex]{voss}
\documentclass{ctexart}% needs xelatex
\pagestyle{empty}
\begin{document}
美好的一天.
\end{document}
\end{externalDocument}
This is made [...]
```

Any change in this example will automatically be kept in sync — during the next translation process, the output of the main document will be updated, and with it the new code of the example will be run, and thus also the new output will be inserted.

In the example above, only the output was included without showing the source code. Depending on the application, it may be desirable to display portions or all of the source code; this is described on the following pages.

Currently the hvextern package supports external documents for METAPOST, TeX, ConTeXt, LaTeX, LuaTeX, LuaLaTeX, XeTeX, XeLaTeX, Lua, Perl, Java, Python, and shell scripts.

2 Syntax

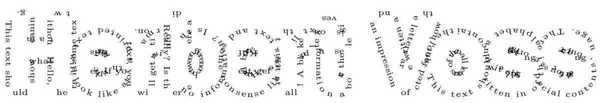
The package, which has no special options, is loaded as usual: `\usepackage{hvextern}`. The package defines only one environment, `{externalDocument}`, and one command, `\runExtCmd`:

Syntax
```
\begin{externalDocument}[⟨options⟩]
        {⟨output filename without extension⟩}
... source code ...
\end{externalDocument}

\runExtCmd[⟨options⟩]
    {⟨command⟩}
    {⟨output filename without extension⟩}
```

The main document must be compiled with the `-shell-escape` option (or with two dashes, as usual), otherwise no external commands will be run and thus the correct output will not be shown.

lualatex invocation
```
lualatex --shell-escape ⟨latexfile⟩
```

Let's show another example: The following code for character manipulation must be compiled using the program sequence `latex`→`dvips`→`ps2pdf`, because it does not work with other TeX engines. With the environment `externalDocument`, however, you can write the complete code in an external file, specify the necessary process, and embed the result as a PDF. The only important thing is that when creating the graphic, the standard output of the page number is suppressed and any white space is cut off using the crop option. Of course, this does not apply if a complete page is to be included (see page 284).



The code of the above example looks like:

```
dvips example
\begin{externalDocument}[compiler=latex,crop,
  force=false, cleanup={log,aux,ps,dvi},
  grfOptions={width=\linewidth}]{voss}
\documentclass{article}
\usepackage[american]{babel}
\pagestyle{empty}
\usepackage{pst-text,blindtext}
\begin{document}
\DeclareFixedFont{\SF}{T1}{phv}{b}{n}{2cm}
\pstextpath(0,-1ex){\pscharpath*[
  linestyle=none]{\SF Herbert Voss}}{%
\tiny \blindtext}
\end{document}
\end{externalDocument}
```

The meaning of each option:

`compiler=latex` Use LaTeX to compile. The rest of the invocation, including other programs, is determined by the internal definition of `\hv@extern@runLATEX`.

`crop` Crop the whitespace with `pdfcrop`.

`force` Recreate the output even if it already exists.

`cleanup={log,aux,ps,dvi}` Delete the specified auxiliary files at the end.

`grfOptions={width=\linewidth}` Scale output to the current linewidth.

`voss` Filename that is extended internally by a consecutive number.

The external filename, extended by a consecutive number, can be printed into the margin by setting the keyword `showFilename`. In general it is printed in the outer margin, or in twocolumn mode in the outer column. If the example is set in twocolumn mode but inside a starred floating environment over both columns, then use the keyword `outerFN` (see Figure 1). Then `hvextern` doesn't test for twocolumn mode.

A vertical shift of the filename is possible by specifying a length for `shiftFN`, e.g., `shiftFN=5ex`.

Essentially, it doesn't matter which programming language is used, as long as minimum communication between the main document and the external program is guaranteed: this consists only of the requirement that the external document must provide its output with the same file name with which it was called. However, even this can be a problem in some programming languages, as shown below with some examples.

By default, source code and output are displayed one above the other, so that a page break in the source code is not a problem. The following example creates and runs a Python program, and then includes the output as a PNG format file. The header of the `externalDocument` environment is:

```
Options for Python
\begin{externalDocument}[compiler=python3,
  code, ext=py, docType=py, usefancyvrb,
  grfOptions={width=\linewidth}]{voss}
... Python code ...
\end{externalDocument}
```

It is only in rare cases that you will want to output the complete source code. Therefore, areas can be defined using so-called markers, which then delimit the output. The markers are written to the external file as normal comments, the only reason why they are programming language dependent; the comment character is not uniform. For Python the markers are:

```
Python marker lines
\hv@extern@ExampleType{py}
  {\NumChar StartVisibleMain}
  {\NumChar StopVisibleMain}
  {\NumChar StartVisiblePreamble}
  {\NumChar StopVisiblePreamble}
```

and for plain TeX:

```
TeX marker lines
\hv@extern@ExampleType{tex}
  {\perCent StartBody}
  {\string\bye}
  {\perCent StartVisiblePreamble}
  {\perCent StopVisiblePreamble}
```

`\perCent` and `\NumChar` are the TeX and Python comment characters % and #, which must be escaped for LaTeX. Internally, the category of the character is changed so that it is available as a normal character using the `\perCent` or `\NumChar` command.

After calling the Python program, it must be ensured that the file name is determined in order to provide the output with the same main file name and a different file extension. In Python this is possible with the following code:

Python: get filename

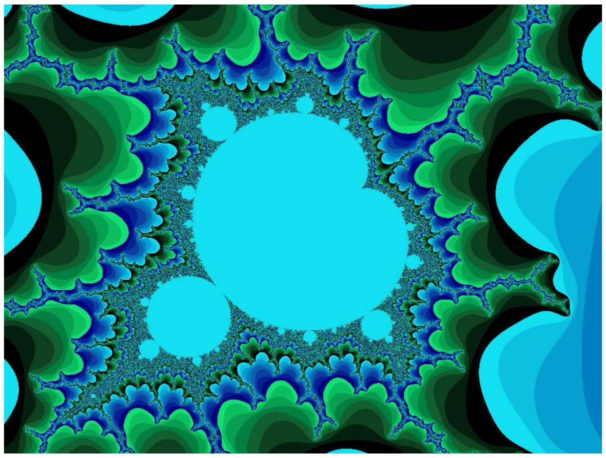
```
fileName = os.path.basename(
   os.path.splitext(__file__)[0])
```

Depending on the output format, this file name is extended by `.pdf`, `.png`, or `.txt`, so that the output can be easily inserted into the LaTeX main document. In addition, the markers are now used so that the output of parts of the Python source code can be done, requested with the `code` keyword. (Output grayscaled for printed *TUGboat*.)

voss-3.py

```
from PIL import Image
import subprocess
# drawing area (xa < xb and ya < yb)
xa = -0.1716
xb = -0.1433
ya = 1.022
yb = 1.044
maxIt = 1024  # iterations
imgx = 1000   # image size
imgy = 750
image = Image.new("RGB", (imgx, imgy))
```

```
for y in range(imgy):
    cy = y * (yb - ya) / (imgy - 1)  + ya
    for x in range(imgx):
        cx = x * (xb - xa) / (imgx - 1) + xa
        c = complex(cx, cy)
        z = 0
        for i in range(maxIt):
            if abs(z) > 2.0: break
            z = z * z + c
        r = i % 4 * 6
        g = i % 8 * 32
        b = i % 16 * 16
        image.putpixel((x, y), b*65536 + g*256 + r)
```



In a purely formal way, the output of the source code can be defined by analogy to LaTeX as a preamble (general definitions) and program body (application), whereby two slightly different background colors are used for differentiation. The markers can be used anywhere in the document. The above example was created with the following LaTeX code:

Complete example code

```
\begin{externalDocument}[compiler=python3, force=false,
  %showFilename,
  runs=1, code, ext=py, docType=py,
  usefancyvrb, grfOptions={width=\linewidth}]{python}
import os
#StartVisiblePreamble
from PIL import Image
import subprocess
# drawing area (xa < xb and ya < yb)
xa = -0.1716; xb = -0.1433
ya = 1.022;   yb = 1.044
maxIt = 1024  # iterations
imgx = 1000   # image size
imgy = 750
image = Image.new("RGB", (imgx, imgy))
#StopVisiblePreamble

#StartVisibleMain
for y in range(imgy):
    cy = y * (yb - ya) / (imgy - 1)  + ya
    for x in range(imgx):
        cx = x * (xb - xa) / (imgx - 1) + xa
        c = complex(cx, cy)
        z = 0
        for i in range(maxIt):
            if abs(z) > 2.0: break
            z = z * z + c
        r = i % 4 * 6
        g = i % 8 * 32
        b = i % 16 * 16
        image.putpixel((x, y), b*65536 + g*256 + r)
#StopVisibleMain
# now get the filename created by the latex
imageName = os.path.basename(
  os.path.splitext(__file__)[0])
image.save(imageName+".png", "PNG")
\end{externalDocument}
```

By specifying a width for the output of the source code, code and result can be arranged side by side, as shown in Figure 1.

3 Using markers in the source code

The markers identify the areas of the source code that are to be output in the (LaTeX) main document. For an external document with TeX or LaTeX code, the use of the markers are shown in the following examples:

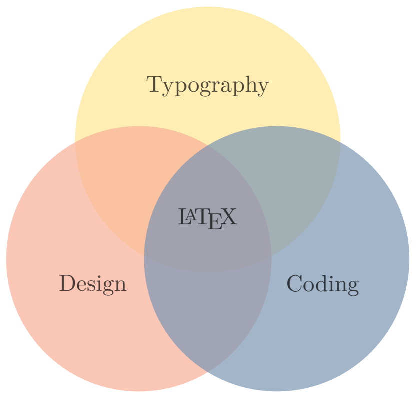
LaTeX marker lines

```
[...]
%StartVisiblePreamble
[... listed preamble code ...]
%StopVisiblePreamble
[...]
\begin{document}
[... listed body code ...]
\end{document}
```

Everything between the `%StartVisiblePreamble` and `%StopVisiblePreamble` lines is printed with the background color `BGpreamble` (default `black!12`). All of the lines between `\begin{document}` and `\end{document}`, on the other hand, are considered as the text body and printed with the background color `BGbody` (default `black!8`).

```
\usepackage{tikz}
\usepackage[hks,pantone,xcolor]{xespotcolor}
```

```
\SetPageColorSpace{HKS}
\definecolor{HYellow}{spotcolor}{HKS05N,0.5}
\definecolor{HRed}{spotcolor}{HKS14N,0.5}
\definecolor{HBlue}{spotcolor}{HKS38N,0.5}
\begin{tikzpicture}[fill opacity=0.7]
 \fill[HYellow]( 90:1.2) circle (2);
 \fill[HRed]   (210:1.2) circle (2);
 \fill[HBlue]  (330:1.2) circle (2);
 \node at ( 90:2)  {Typography};
 \node at ( 210:2) {Design};
 \node at ( 330:2) {Coding};
 \node {\LaTeX};
\end{tikzpicture}
```



voss-4.tex

Figure 1: Example for side-by-side code and output inside a `figure*` environment in twocolumn mode.

For TeX we use:

```
──────── TeX marker lines ────────
[...]
%StartVisiblePreamble
[... listed preamble code ...]
%StopVisiblePreamble
[...]
%StartBody
[...]
\bye
```

Now everything between `%StartBody` and `\bye` is the printed text body.

The markers are defined by the internal macro `\hv@extern@ExampleType`. This macro expects five parameters, for example for Java:

```
──────── Java marker lines ────────
\hv@extern@ExampleType{java}
  {//StartVisibleMain}
  {//StopVisibleMain}
  {//StartVisiblePreamble}
  {//StopVisiblePreamble}
```

The comment starter for Java is `//`, for Lua `--`, and for Perl `#`. The latter must be escaped by using `\NumChar`, as already shown in an example above. In general, the option `docType` defines the type of the source code (the comment starter), and it must have one of these values:

`context java latex lua mp pl py tex sh`

As you can see, only `tex` is allowed for `docType`, not `latex`, `pdflatex`, etc. This is because the comment starter is uniformly `%` for all TeX variants.

The `compiler` option defines the base program to be run, and the entire invocation, which may involve additional programs. The following `compiler` values are currently supported:

`context java latex lua lualatex luatex mpost pdflatex perl python3 sh tex texlua xelatex xetex`

The configurations for Lua, Perl, Java, shell, and Python all have the same structure; they only differ in the comment character to be used. For Lua, we have

```
──────── Lua marker lines ────────
\hv@extern@ExampleType{lua}
  {--StartVisibleMain}
  {--StopVisibleMain}
  {--StartVisiblePreamble}
  {--StopVisiblePreamble}
```

Sometimes, both `docType` and `compiler` are the same, for example when using Lua: `docType=lua` and `compiler=lua`. Indeed, for Lua files that also have the `.lua` extension, the value `lua` must be assigned three times:

```
──────── Options for Lua code ────────
ext=lua, compiler=lua, docType=lua,
```

The following Lua example writes plain text to standard output, so we pass the `redirect` option to the `externalDocument` environment; the output is then redirected into a file of the same main name but with the extension `.txt`. This is read verbatim from within the main LaTeX document and can therefore contain any characters.

```
io.write("1. "..type("Hello world").."  ")
print("2. "..type(10.4*3))
io.write("3. "..type(print).."  ")
io.write("4. "..type(type).."  ")
print("5. "..type(true))
io.write("6. "..type(nil).."  ")
print("7. "..type(type(X)))
```

voss-5.lua

```
io.write("8. "..type(a).."  ")

a = 10
io.write("9. "..type(a).."  ")
a = "a string!!"
io.write("10. "..type(a).."  ")
a = print
print("11. "..type(a))
```

```
1. string
2. number
3. function
4. function
5. boolean
6. nil
7. string
8. nil
9. number
10. string
11. function
```

The same applies to the following example with Java code:

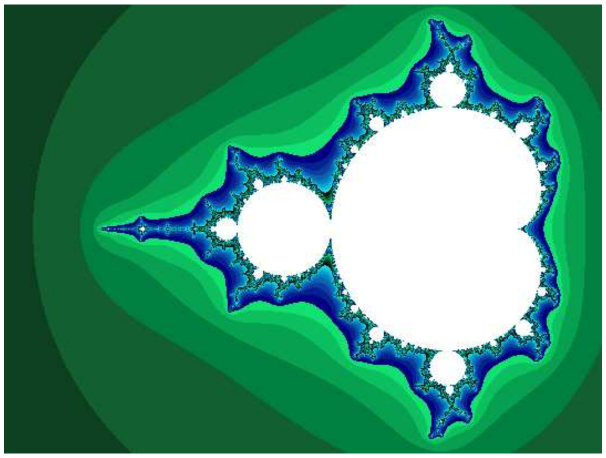
Options for Java code

```
ext=java, compiler=java, docType=java,
```

voss-6.java

```
  public static int iterZahl(
    final double cx,
    final double cy,
    int maxIt,
    final double radius){
  // count the number of iterations
    int zaehler = 0;
    double zx = 0.0, zy = 0.0, tmp;
    do {
      tmp = zx*zx - zy*zy + cx;
      zy = 2*zx*zy + cy;
      zx = tmp;
      zaehler++;
  // run as long as the kength of the vector
  // is smaller than the radius
    } while (zx*zx+zy*zy<=radius && zaehler<maxIt);
    return zaehler;
  }
```

```
    double xa = -2.5, xe = 0.7, ya = -1.2, ye = 1.2;
    double dx = (xe-xa)/(imageWidth-1),
           dy = (ye-ya)/(imageHeight-1);
    double cx, cy; int R, G, B;
    double radius = 10.0;  int maxIt = 1024;
    cx = xa;
    for (int sp = 0; sp < imageWidth; sp++) {
      // from top to bottom:
      cy = ye;
      for (int ze = 0; ze < imageHeight; ze++) {
        int zIter = iterZahl(cx,cy,maxIt,radius);
        if (zIter == maxIt) {
          g.setColor(Color.WHITE);
          g.drawLine(sp, ze, sp, ze);
        } else {
          R = zIter % 4 * 6 ;
          G = zIter % 8 * 32;
          B = zIter % 16 * 16;
          g.setColor(new Color(R,G,B));
          g.drawLine(sp, ze, sp, ze);
        }
        cy = cy - dy;
      } // for ze
      cx = cx + dx;
    } // for sp
```



4 Options

4.1 Program(s) and number of runs

In general, any selected `compiler` program should be found in your search path, with `pdflatex` being the default. However, in rare cases it may be necessary to specify a path for the program, which is done by assigning to `progpath`. A `/` must appear at the end, for example ‘`progpath=./bin/`’.

Here is the code defining the options `progpath`, `compiler`, `runs`, and `runsequence`. The full list of `compiler` values was given on previous page. (The definitions are omitted.)

Compiler options

```
\define@key{hv}{progpath}[]{...}
\define@choicekey*+{hv}{compiler}[\val\nr]{%
    context,...,xetex}
    [pdflatex]{...}
\define@key{hv}{runs}[1]{...}
\define@key{hv}{runsequence}[]{...}
```

Instead of using `compiler`, `biber` and `xindex`, an explicit run sequence can also be specified via the `runsequence` parameter. A comma-separated list is expected. The input filename is added to each program being run. For example, this sequence generates the bibliography and (with additional options) index, besides the main document:

Invocation (runsequence) example

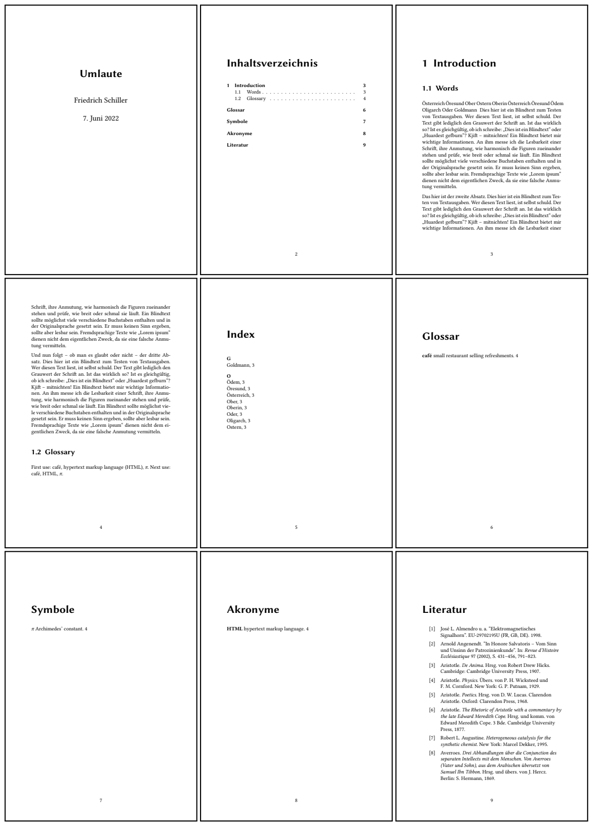
```
runsequence={lualatex,biber,xindex -l de -c DIN2,
  makeglossaries,lualatex,lualatex},
cleanup={log, aux, toc, bbl, blg,
  run.xml, bcf, idx, ilg},
pages={1,2,3,4,5,6,7,8,9},
```

The example also prints pages 1–9 of the created external document, which also has a glossary and a list of symbols and acronyms.

voss-7.tex

```
\documentclass[paper=a5,parskip=half-,DIV=12,
     bibliography=totoc,
     listof=totoc,fontsize=12pt]{scrreprt}
\usepackage[ngerman]{babel}
\usepackage{libertinus-otf,hvindex}
\usepackage{biblatex,makeidx}\makeindex
\addbibresource{biblatex-examples.bib}
\usepackage[abbreviations,symbols,postdot,
    stylemods,style=index]{glossaries-extra}
\makeglossaries
\title{Umlaute} \author{Friedrich Schiller}
```

```
\maketitle \tableofcontents
\chapter{Introduction}   \section{Words}
\Index{Österreich} \Index{Öresund}
\Index{Ober} \Index{Ostern} \Index{Oberin}
\Index{Österreich} \Index{Öresund}
\Index{Ödem} \Index{Oligarch} \Index{Oder}
\Index{Goldmann}  \Blindtext[3]
\section{Glossary}
First use: \gls{cafe}, \gls{html}, \gls{pi}.
Next use: \gls{cafe}, \gls{html}, \gls{pi}.
\printindex    \printglossaries
\nocite{*}\raggedright\printbibliography
```



Umlaute

Friedrich Schiller

7. Juni 2022

Inhaltsverzeichnis

1 **Introduction** 3
1.1 Words 3
1.2 Glossary 4
Glossar 6
Symbole 7
Akronyme 8
Literatur 9

2

1 Introduction

1.1 Words

Österreich Öresund Ober Ostern Oberin Österreich Öresund Ödem Oligarch Oder Goldmann Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext" oder „Huardest gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum" dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Das hier ist der zweite Absatz. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext" oder „Huardest gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer

3

Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum" dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Und nun folgt – ob man es glaubt oder nicht – der dritte Absatz. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext" oder „Huardest gefburn"? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum" dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

1.2 Glossary

First use: café, hypertext markup language (HTML), π. Next use: café, HTML, π.

4

Index

G
Goldmann, 3

O
Ödem, 3
Öresund, 3
Österreich, 3
Ober, 3
Oberin, 3
Oder, 3
Oligarch, 3
Ostern, 3

5

Glossar

café small restaurant selling refreshments. 4

6

Symbole

π Archimedes' constant. 4

7

Akronyme

HTML hypertext markup language. 4

8

Literatur

[1] José L. Almendro u. a. "Elektromagnetisches Signalhorn". EU-29702195U (FR, GB, DE). 1998.
[2] Arnold Angenendt. "In Honore Salvatoris – Vom Sinn und Unsinn der Patrozinienkunde". In: *Revue d'Histoire Ecclésiastique* 97 (2002), S. 431–456, 791–823.
[3] Aristotle. *De Anima*. Hrsg. von Robert Drew Hicks. Cambridge: Cambridge University Press, 1907.
[4] Aristotle. *Physics*. Übers. von P. H. Wicksteed und F. M. Cornford. New York: G. P. Putnam, 1929.
[5] Aristotle. *Poetics*. Hrsg. von D. W. Lucas. Clarendon Aristotle. Oxford: Clarendon Press, 1968.
[6] Aristotle. *The Rhetoric of Aristotle with a commentary by the late Edward Meredith Cope*. Hrsg. und komm. von Edward Meredith Cope. 3 Bde. Cambridge University Press, 1877.
[7] Robert L. Augustine. *Heterogeneous catalysis for the synthetic chemist*. New York: Marcel Dekker, 1995.
[8] Averroes. *Drei Abhandlungen über die Conjunction des separaten Intellects mit dem Menschen. Von Averroes (Vater und Sohn), aus dem Arabischen übersetzt von Samuel Ibn Tibbon*. Hrsg. und übers. von J. Hercz. Berlin: S. Hermann, 1869.

9

4.2 Graphics options

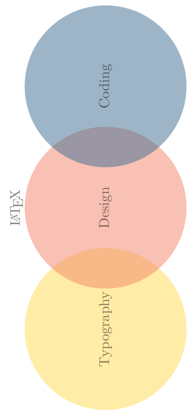
```
──────────── Graphics options ────────────
\define@key{hv}{grfOptions}[]{...}
```

The value for `grfOptions` is passed to the well-known `\includegraphics` macro, for example `{angle=90, width=\linewidth}`, as shown in the following example.

voss-8.tex

```
\usepackage{tikz}
% needs xelatex:
\usepackage[hks,pantone,
  xcolor]{xespotcolor}
```

```
\SetPageColorSpace{HKS}
\definecolor{HYellow}{spotcolor}
  {HKS05N,0.8}
\definecolor{HRed}{spotcolor}
  {HKS14N,0.8}
\definecolor{HBlue}{spotcolor}
  {HKS38N,0.8}
\begin{tikzpicture}
  [fill opacity=0.5]
\fill[HYellow](0,0)circle(2);
\fill[HRed]   (3,0)circle(2);
\fill[HBlue]  (6,0)circle(2);
 \node at (0,0){Typography};
 \node at (3,0){Design};
 \node at (6,0){Coding};
 \node at (3,2.2){\LaTeX};
\end{tikzpicture}
```



Since source code and output are arranged next to each other here, the specification `\linewidth` refers to the current width of the minipage. Thus the output has the maximum possible size.

4.3 Listing options

```
──────────── Listings options ────────────
\define@key{hv}{lstOptions}[]{...}
```

The value assigned is passed to either the macro `\lstinputlisting` or, if `usefancyvrb` is specified, to the macro `\VerbatimInput` from the `fancyvrb` package. It should be noted that the options for the respective packages have different meanings and names, so it is not so easy to switch between `listings` and `fancyvrb`.

The following example uses the `listings` package, which is the default and therefore does not require any parameter setting. A slightly exotic list of options is given, and we omit the graphical output (which is seen in the next example), purely for the demonstration:

```
──────────── Listings options example ────────────
lstOptions={basicstyle=\sffamily\slshape\scriptsize,
             columns=fullflexible,showoutput=false},
```

voss-9.tex

```
\documentclass[landscape]{article}
\usepackage[margin=1cm]{geometry}
\usepackage{pst−calendar}
```

```
\psscalebox{0.5}{%
  \psCalDodecaeder[
    Year=2022,style=june]%
}
\hspace{4cm}
\psscalebox{0.5}{%
  \psCalDodecaeder[
    Year=2022,style=july]%
}
```

4.4 Background color for the code

Different colors for the background and the frame can be selected. They can be modified via the following parameters, which show the defaults in brackets. (The actual definitions are omitted.) `BG` is the abbreviation for “background” and `BO` for “border”:

```
Color options
\define@key{hv}{BGpreamble}[black12]{...}
\define@key{hv}{BGbody}[black8]{...}
\define@key{hv}{BOpreamble}[black12]{...}
\define@key{hv}{BObody}[black8]{...}
```

The parameter values are passed to a `tcolorbox` environment (of the package with the same name), and evaluated there. [3] Because the background and frame have the same color, the frame remains “invisible” by default. This changes with different values, for example:

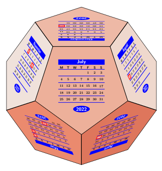
```
Differing frame and background colors
  BGpreamble=red!10, BOpreamble=red,
  BGbody=blue!8, BObody=blue,
```

Typically, you should use subtle colors so that the output does not fade into the background compared to the code.

voss-10.tex

```
\usepackage{pst-calendar}
```

```
\psscalebox{0.3}{%
  \psCalDodecaeder[
    Year=2022,style=july]%
}
```



We’ll return to the default gray colors now.

4.5 Type of source code

The current version of `hvextern` supports source code in METAPOST, plain TEX, LATEX, ConTEXt, Python, Lua, shell, and Perl. Each language’s definition contains the source code markers already mentioned, and the program invocation sequence if special treatment is necessary. For example, source code in LATEX requires special treatment if the program used is `latex`; the corresponding definition contains the following:

```
Marker and run setting for dvips
\hv@extern@ExampleType{latex}
  % for _all_ LaTeX engines
  {\string\begin\string{document\string}}
  {\string\end\string{document\string}}
  {\perCent StartVisiblePreamble}
  {\perCent StopVisiblePreamble}

% only for the sequence latex->dvips->ps2pdf
\def\hv@extern@runLATEX#1#2#3#4{%
  %path/compiler/file/extension
  \ShellEscape{#1#2\space #3#4}%
  \ShellEscape{#1dvips\space #3.dvi}%
  \ShellEscape{#1ps2pdf\space
     -dAutoRotatePages=/None\space
     -dALLOWPSTRANSPARENCY\space#3.ps}%
}
```

The macro must have the following structure:

```
Macro implementing the run sequence
\def\hv@extern@run<NAME>#1#2#3#4{%
  %path/compiler/file/extension
  ...}
```

The definition for TEX is similar. The type of source code and the program used can be different for TEX, LATEX and ConTEXt, for example type `latex` but program `lualatex`.

4.6 Output of one or more full pages

In the event that only a subset of pages are to be output, this can be controlled via the `pages` parameter, as we saw in a previous example. It expects a comma-separated list of the pages to be printed. The individual pages can be framed with the `frame` parameter in order to achieve a clearer presentation.

```
Output page selection
\define@key{hv}{pages}[1]{...}
\define@key{hv}{pagesep}[1em]{%
     \hv@extern@pagesep=#1}
\define@boolkey{hv}[hv@extern@]{frame}[true]{}
```

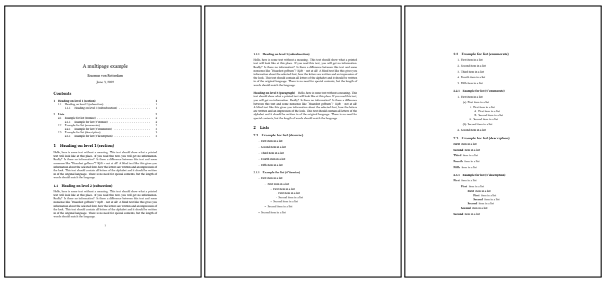
It is up to the user to use the `grfOptions` parameter to ensure that the pages for the output are scaled as needed. This example outputs the first three pages of a document:

```
Page selection example
pages={1,2,3}, grfOptions={width=0.3\linewidth},
pagesep=1pt,
frame, compiler=lualatex, runs=2, % for the TOC
```

voss-11.tex

```
\usepackage[american]{babel}
\usepackage{libertinus}
\usepackage{blindtext}
```

```
\title{A multipage example}
\author{Erasmus von Rotterdam}
\maketitle
\tableofcontents
\blinddocument
```



4.7 Output as a float

As a rule, the output is in the running text, which can be undesirable if the text width is relatively small. Larger free spaces can then arise on one side, which is always unfavorable. In such cases you should use the `float` option, in which case, as usual, a caption can be specified using the `caption` parameter and a cross-reference label can be specified using `label`. The floating type is by default `figure` and the placement can be set by the optional argument `floatsetting`. It is preset to `!htb`.

voss-12.tex

```
\usepackage{pst-coxeterp}
```

```
\begin{pspicture}(-1,-1)(1,1)
  \Simplex[dimension=2]\end{pspicture}
\begin{pspicture}(-1,-1)(1,1)
  \Simplex[dimension=3]\end{pspicture}
\begin{pspicture}(-1,-1)(1,1)
  \Simplex[dimension=5]\end{pspicture}
\begin{pspicture}(-1,-1)(1,1)
  \Simplex[dimension=7]\end{pspicture}
```

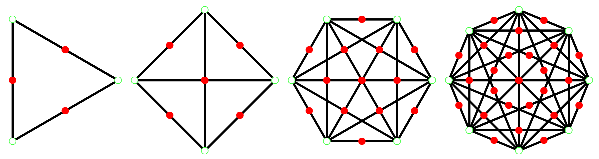


Figure 2: An example for Coxeter images.

```
Float options
\define@boolkey{hv}[hv@extern@]{float}[true]{}
\define@key{hv}{caption}[]{...}
\define@key{hv}{label}[]{...}
```

Figure 2 shows an example as a floating object. It was created with the following parameters:

```
Float example
[...]
 float,
 caption={An example for Coxeter images.},
 label=img:cox,
[...]
```

The specification `float` refers only to the output; otherwise, a previous listing could not have a page break and the typesetting of the text would be more difficult. On the other hand, it may well be that other text appears between the code and the output of an example. Then manual intervention is necessary, for example by using the command `\captionof` from the package `caption`, which allows a caption without floating space.

5 Cropping white space

When displaying the output of examples, usually only the area that contains a graphic or text is of interest, and we want to remove any surrounding white space. For documents that consist of only one page, the document class `standalone` can generally be used, which automatically removes all white space. If you have more than one page or want to use another special document class, `hvextern` provides the `crop` option:

```
Crop options
\define@boolkey{hv}[hv@extern@]{crop}[true]{}
\define@key{hv}{cropmargin}[2]{...}% in pt
```

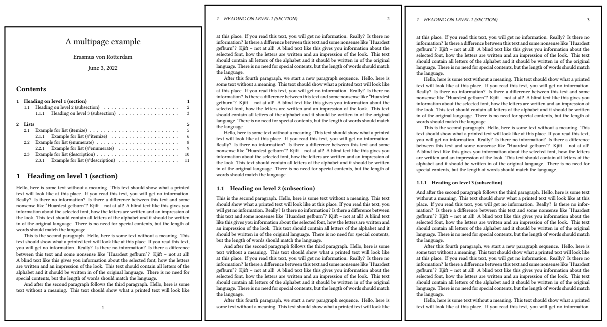
`crop` can also be applied to documents with multiple pages. In this case, however, you should make sure that the pages have headers and footers so that the white space that is cut off always has the same size. Otherwise the pages end up with different heights, as shown in the example below, which is usually undesirable. Among other things, the following parameters were set:

```
Crop example
pages={1,2,3}, grfOptions={width=0.3\linewidth},
frames, pagesep=1pt, crop, cropmargin=5,%is 5pt
compiler=lualatex, runs=2, % for the TOC
```

voss-13.tex

```
\usepackage[american]{babel}
\usepackage{libertinus}
\usepackage{blindtext}
\pagestyle{headings}
```

```
\title{A multipage example}
\author{Erasmus von Rotterdam}
\maketitle
\tableofcontents
\Blinddocument
```



A multipage example

Erasmus von Rotterdam

June 3, 2022

Contents

1	**Heading on level 1 (section)**	**1**
1.1	Heading on level 2 (subsection)	2
1.1.1	Heading on level 3 (subsection)	3
2	**Lists**	**5**
2.1	Example for list (itemize)	5
2.1.1	Example for list (4*itemize)	6
2.2	Example for list (enumerate)	8
2.2.1	Example for list (4*enumerate)	9
2.3	Example for list (description)	10
2.3.1	Example for list (4*description)	11

1 Heading on level 1 (section)

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like

1

1 HEADING ON LEVEL 1 (SECTION) 2

at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

1.1 Heading on level 2 (subsection)

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like

1 HEADING ON LEVEL 1 (SECTION) 3

at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

1.1.1 Heading on level 3 (subsection)

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

After this fourth paragraph, we start a new paragraph sequence. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information.

5.1 Source code and output side by side

Normally the source code is printed first and then the output. This order cannot be changed with the current version of `hvextern`. For side-by-side output, the `mpwidth` parameter determines the width of a left minipage and is always evaluated if it is greater than 0 pt. A second minipage is then reserved for output for the remainder of the line, except for the value of `mpsep`. Both minipages are aligned to the value of `mpvalign`, the top edge by default.

```
———————— Side-by-side options ————————
\define@key{hv}{mpwidth}[0pt]{...}
\define@key{hv}{mpsep}[1em]{...}
```

The default distance between the two minipages is 1 em, as shown.

5.2 Horizontal alignment of the output

The code is always left-aligned, whereas the output can use various known alignments via the `align` option. The use of the `ragged2e` package does not have any advantages here.

```
———————— Horizontal alignment ————————
\define@key{hv}{align}[\centering]{...}
```

The default with `align=\centering`:

```
\rule{0.5\linewidth}{3mm}
```

Left-justified with `align=\raggedright`:

```
\rule{0.5\linewidth}{3mm}
```

Right-justified with `align=\raggedleft`:

```
\rule{0.5\linewidth}{3mm}
```

Side by side, default with `align=\centering`:

```
\rule{0.2\linewidth}{3mm}
```

Side by side, with `align=\raggedright`:

```
\rule{0.2\linewidth}{3mm}
```

Side by side, with `align=\raggedleft`:

```
\rule{0.2\linewidth}{3mm}
```

5.3 Inline output, rather than displayed

The output can be printed within a line in the so-called inline mode. This can make sense if you don't have certain characters available in your document's font, but they can be generated externally and then input as PDF. Here, the corresponding source code should not be shown, so `code=false` is automatically set with `inline`.

```
———————— Inline option ————————
\define@boolkey{hv}[hv@extern@]{inline}[true]{...}
```

An example has already been shown on page 280.

5.4 Handling plain text output

For LaTeX documents, the output is generally PDF, but when using a programming language such as Perl, the output would more typically be plain text. This must be redirected or written to a file so that it can be inserted verbatim into the main document. This can be controlled with the parameters `includegraphic` and `redirect`. The output is typeset with `listings` or `fancyvrb`, and options for the typesetting environment set with `textoptions`.

With `includegraphic=false` it is up to the user to ensure that every output within the external document is written to a text file; `hvextern` looks for a file with the right name. This is done automatically with `redirect`, but then all program output ends up in the external text file.

```
———————— Plain text output options ————————
\define@boolkey{hv}[hv@extern@]{redirect}[true]{}
\define@boolkey{hv}{includegraphic}[true]{}
\define@key{hv}{textOptions}[]{...}
```

The text file must have the same main file name as the external file, but with the extension `.txt`. As we've seen, each program can determine by itself what name it was called with, so it is easily possible to determine the correct name for the text output file. For a Perl program, this could be achieved with the following code, for example:

```
———————— Perl: get filename ————————
my $filename = $0; # the current filename
$filename =~ s/\.pl//; # without extension .pl
$filename = "${filename}.txt"; # for the output
open(my $fh, '>', $filename);
```

However, in the next example, the optional keyword `redirect` is given, so determining the filename in the code is not needed. The example is set with:

```
———————— Example output redirect ————————
compiler=perl, includegraphic=false, docType=pl,
ext=pl, usefancyvrb, runs=1, code, redirect,
tcbox=false, force, lstOptions={fontsize=\small,
  fontfamily=tt, frame=lines}
```

```
my $number = 1;
my $start = 1;
my $end    = 9;
my $found = 0;
```

voss-20.pl

```
print "Searching for Kaprekar constants\n";
while ($number < 8) {
 print "${number}-digits: ";
 foreach ($start...$end){
  @chars = split(//,$_);
  $Min = join("",sort(@chars));
  $Max = reverse($Min);
  $Dif=$Max-$Min;
  if($_ eq $Dif) {
   $found = 1;
   print $_,",  ";
  }
 }
 if (!$found) { print "---\n"; }
 else         { print "\n"; }
 $found = 0;
 $number++;
 $start = $start*10;
 $end   = $end*10;
}
```

```
Searching for Kaprekar constants
1-digits: ---
2-digits: ---
3-digits: 495,
4-digits: 6174,
5-digits: ---
6-digits: 549945,  631764,
7-digits: ---
```

(Just for the sake of completeness: A Kaprekar constant is a number A with $\max(A) - \min(A) = A$, where max and min are the sorted/reverse-sorted digits of the number, e.g., $A = 495 = 954 - 459$.)

Our next example is in Lua, and also produces text output; but instead of using `redirect`, the code outputs to the appropriate file. This filename can be determined as follows:

```
Lua: get filename
-- get full filename
local filename = arg[0]
-- delete extension
local shortFN = str:match("(.+)%..+")
-- open external file
outFile = io.open(shortFN..".txt","w+")
```

```
function nextrow(t)
  -- fill table
  local ret = {}
  t[0], t[#t+1] = 0, 0
  for i = 1, #t do
    ret[i]=t[i-1]+t[i]
  end
  return ret
end

function triangle(n)
 t = {1}
 for i = 1, n do
  m = (n - i)
  for j = 1,m do
    outFile:write("  ")
  end
  for k = 1,i do
   outFile:write(
    string.format("%4s",t[k]))
  end
  outFile:write("\n")
  t = nextrow(t)
 end
end
```

voss-21.lua

```
triangle(10)
```

```
                     1
                   1   1
                 1   2   1
               1   3   3   1
             1   4   6   4   1
           1   5  10  10   5   1
         1   6  15  20  15   6   1
       1   7  21  35  35  21   7   1
     1   8  28  56  70  56  28   8   1
   1   9  36  84 126 126  84  36   9   1
```

5.5 Generating the bibliography and index

The current version of `hvextern` has predefined support for constructing the bibliography with Biber and an index with `xindex`, via the following parameters:

```
Index and bibliography options
\define@boolkey{hv}[hv@extern@]{biber}[true]{}
\define@boolkey{hv}{xindex}[true]{}
\define@key{hv}{xindexOptions}[]{...}
```

In principle, the external run of Biber does not require any further parameters, whereas the `xindex` program requires information about the language, the style file, etc., for example. The next example is generated with the following parameters:

```
xindex example
\begin{externalDocument}[
compiler=lualatex,runs=2,pages=2,crop,
xindex,xindexOptions={-l DE --config DIN2},
docType=latex,cleanup={log,aux,ilg,idx},...]
```

To use other bibliography or index programs, you can use the `runsequence` option; see the example on p. 284.

```
\usepackage{makeidx}
\makeindex
\usepackage{hvindex}
```

```
\Index{Österreich} \Index{Öresund}
\Index{Ödipus} \Index{Öchsle}
\Index{Ostern} \Index{Ober} \Index{Oberin}
\Index{Österreich} \Index{Öresund}
\Index{Ödem} \Index{Oligarch} \Index{Oder}
\Index{Ostern} \Index{Ober} \Index{Oberin}
\Index{Obstler} \Index{Öl} \Index{ölen}
\Index{Oder|seealso{Fluss}} \Index{Göbel}
\Index{oder} \index{Fluss!Oder}
\Index{Goethe} \Index{Göthe} \Index{Götz}
\Index{Goldmann}
\printindex
```

voss-22.tex

Index

F
Fluss
- Oder, 1

G
Göbel, 1
Göthe, 1
Goethe, 1
Götz, 1
Goldmann, 1

O
Ödem, 1
Öl, 1
ölen, 1
Öresund, 1
Österreich, 1
Ober, 1
Oberin, 1
Obstler, 1
Oder, 1
oder, 1
Oder, *siehe auch* Fluss
Oligarch, 1
Ostern, 1

6 Verbatim modes: `listings` and `fancyvrb`

6.1 Using `listings`

By default the command `\lstinputlisting` from the package `listings` is used for printing the source code. We saw an example of setting `hvextern`'s `lstOptions` option for it earlier.

6.2 Package `fancyvrb`

There are no fundamental objections to the `listings` package, but sometimes it makes more sense to use `\VerbatimInput` from the `fancyvrb` package, especially when displaying non-ASCII Unicode characters. Most of the examples in this article use `fancyvrb`, by passing the `usefancyvrb` option.

6.3 Vertical space

Vertical space can be controlled by four keywords for the stretchable vertical space:

`aboveskip` The vertical space before the environment `externalDocument` or the command `\runExtCmd` (default `\medskipamount`).

`belowpreambleskip` The vertical space between the preamble and body (default `\smallskipamount`). If the preamble is missing, then there will be only `aboveskip`.

`belowbodyskip` The vertical space between body and output (default `\smallskipamount`).

`belowskip` The vertical space after the environment `externalDocument` or the command `\runExtCmd` (default `\medskipamount`).

7 Supported METAPOST and TEX engines

Here we show a few examples using the common TEX-world programs. (LATEX is omitted here since most of the examples throughout this article use LATEX.)

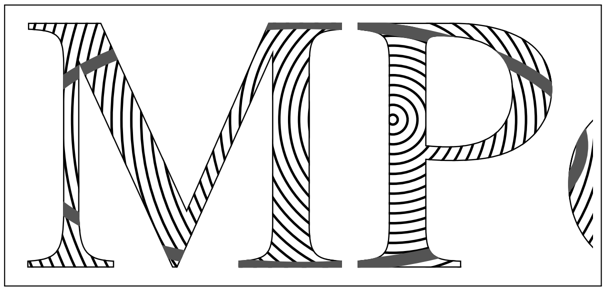
7.1 METAPOST example

```
MetaPost example
\begin{externalDocument}[
  compiler=mpost,docType=mp,...]
```

```
defaultfont:="ptmr8r";
warningcheck:=0;
```

```
draw fullcircle shifted (0.5,0.6) xscaled 8cm
  yscaled 3.5cm withpen pencircle scaled 5bp
  withcolor 0.33; % gray bands
special("/Times-Roman findfont 150 "
& " scalefont setfont "
& " 0 10 moveto (MPost) false charpath 2 "
& " clip stroke gsave 150 70 translate "
& " 2 4 600 {dup 0 moveto 0 exch 0 exch"
& " 0 360 arc stroke} for grestore ");
```

voss-23.mp



Here is the definition of the command sequence for running METAPOST, just in case you want to modify something:

```
MetaPost run sequence
\def\hv@extern@runMP#1#2#3#4{%
  % path / compiler / file / extension
  \ShellEscape{#1#2\space -tex=tex\space #3#4}%
  \ShellEscape{#1tex\space "\string\input\space
    epsf\string\relax\string\nopagenumbers
    \string\epsfbox{#3.1}\string\bye"}%
  \ShellEscape{#1dvips\space -j\space -E\space
    -o\space #3.eps\space epsf.dvi}%
  \ShellEscape{#1epstopdf\space #3.eps}%
}
```

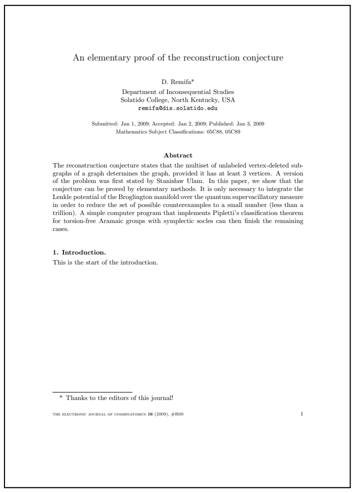
7.2 Plain TEX example

```
Plain TEX example
\begin{externalDocument}[
  compiler=tex,docType=tex,...]
```

voss-24.tex

```
\footline={\footsc the electronic journal
 of combinatorics {\footbf 16} (2009),
 \#R00\hfil\footrm\folio}
```

```
\font\bigrm=cmr12 at 14pt
\centerline{\bigrm An elementary proof
  of the reconstruction conjecture}

\bigskip\bigskip
\centerline{D. Remifa\footnote*{Thanks to the
  editors of this journal!}}
\smallskip
\centerline{Department of Inconsequential Studies}
\centerline{Solatido College, North Kentucky, USA}
\centerline{\tt remifa@dis.solatido.edu}
\bigskip
\centerline{\footrm
Submitted: Jan 1, 2009; Accepted: Jan 2, 2009;
  Published: Jan 3, 2009}
\centerline{\footrm Mathematics Subject
  Classifications: 05C88, 05C89}
\bigskip\bigskip
\centerline{\bf Abstract}
\smallskip
{\narrower\noindent
The reconstruction conjecture states that the
multiset of unlabeled vertex-deleted subgraphs
of a graph determines the graph, provided it
has at least 3 vertices.  A version of the problem
was first stated by Stanis\l aw Ulam. In this
paper, we show that the conjecture can be proved
by elementary methods.  It is only necessary to
integrate the Lenkle potential of the Broglington
manifold over the quantum supervacillatory measure
in order to reduce the set of possible
counterexamples to a small number (less than a
trillion).  A simple computer program that
implements Pipletti's classification theorem for
torsion-free Aramaic groups with symplectic socles
can then finish the remaining cases.}

\bigskip
\beginsection 1. Introduction.

This is the start of the introduction.
```



An elementary proof of the reconstruction conjecture

D. Remifa*
Department of Inconsequential Studies
Solatido College, North Kentucky, USA
remifa@dis.solatido.edu

Submitted: Jan 1, 2009; Accepted: Jan 2, 2009; Published: Jan 3, 2009
Mathematics Subject Classifications: 05C88, 05C89

Abstract

The reconstruction conjecture states that the multiset of unlabeled vertex-deleted subgraphs of a graph determines the graph, provided it has at least 3 vertices. A version of the problem was first stated by Stanisław Ulam. In this paper, we show that the conjecture can be proved by elementary methods. It is only necessary to integrate the Lenkle potential of the Broglington manifold over the quantum supervacillatory measure in order to reduce the set of possible counterexamples to a small number (less than a trillion). A simple computer program that implements Pipletti's classification theorem for torsion-free Aramaic groups with symplectic socles can then finish the remaining cases.

1. Introduction.
This is the start of the introduction.

* Thanks to the editors of this journal!

THE ELECTRONIC JOURNAL OF COMBINATORICS **16** (2009), #R00 1

7.3 ConTEXt example (mkIV)

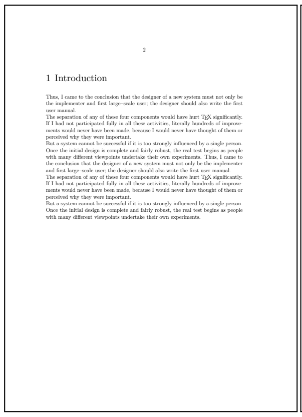
This example is run with ConTEXt from TEX Live 2022, but it should also work with the new LMTX.

```
ConTEXt example
\begin{externalDocument}[pages={3,4},
compiler=context,docType=context,runs=2,...]
```

voss-25.tex

```
\definehead
  [myhead]
  [section]
\setuphead
  [myhead]
  [numberstyle=bold,
   textstyle=bold,
   before=\hairline\blank,
   after=\nowhitespace\hairline]
```

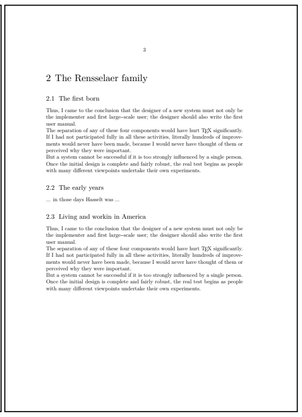
```
\startstandardmakeup
\midaligned{From Hasselt to America}
\midaligned{by}
\midaligned{J. Jonker and C. van Marle}
\stopstandardmakeup
\placecombinedlist[content]
\chapter{Introduction}
\input knuth \input knuth
\chapter[rensselaer]{The Rensselaer family}
\section{The first born}
\input knuth
\section{The early years}
... in those days Hasselt was ...
\section{Living and workin in America}
\input knuth
\chapter[lansing]{The Lansing family}
\input knuth
... the Lansing family was also ...
\chapter[cuyler]{The Cuyler family}
\input knuth
... much later Tydeman Cuyler ...
\myhead[headlines]{And the end}
foo
```



2

1 Introduction

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual.
The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.
But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments. Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual.
The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.
But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.



3

2 The Rensselaer family

2.1 The first born

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual.
The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.
But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

2.2 The early years

... in those days Hasselt was ...

2.3 Living and workin in America

Thus, I came to the conclusion that the designer of a new system must not only be the implementer and first large–scale user; the designer should also write the first user manual.
The separation of any of these four components would have hurt TEX significantly. If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important.
But a system cannot be successful if it is too strongly influenced by a single person. Once the initial design is complete and fairly robust, the real test begins as people with many different viewpoints undertake their own experiments.

8 Running arbitrary external commands

To typeset listing of the current directory of this document we can use the macro `\runExtCmd` with the optional argument `redirect`. We filter the output with additional commands.

```
\runExtCmd[redirect]
  {ls -laB | awk '{print $6,$7,$8,$9}' }
  {voss}
```

to produce the directory listing:

```
Nov 3 18:49 .
Nov 1 23:39 ..
Jun 3 17:36 .dict.pws
Nov 3 18:49 Exa-extern
Jun 3 18:04 Makefile
Nov 3 18:49 firstpage.tex
Nov 3 18:49 lastpage.tex
Nov 3 18:49 tb135voss-extern.aux
Nov 3 18:49 tb135voss-extern.bbl
Jun 3 17:36 tb135voss-extern.bib
Nov 3 18:49 tb135voss-extern.blg
Nov 3 18:49 tb135voss-extern.log
Oct 31 16:12 tb135voss-extern.ltx
Nov 3 18:49 tb135voss-extern.out
Nov 3 18:49 tb135voss-extern.pdf
```

The general behaviour is similar to the environment, `externalDocument`: the output is saved in an intermediate file, in this case `voss-`⟨*num*⟩`.txt` and then read by `\VerbatimInput`. The options `code` and `showFilename` are off by default.

9 Other options

Most of the options which `hvextern` provides for `externalDocument` and `\runExtCommand` have been discussed. Here is a brief summary of some that have not been seen, or mentioned only in passing.

`force` With `force=false` an existing output file is used, thus reducing compilation time. This option should only be used in exceptional cases, because with it, a change in the main document in the source code of the example does not lead to updated example output.

`moveToExampleDir` Move all generated example files, both source and output, to the directory specified by `ExamplesDir`. This can ease document development and maintenance when there are many examples. The directory itself must be created by the user.

`ExampleDir` The directory to which example files are moved if requested.

`cleanup` Auxiliary files from an (LA)TEX or other run can be deleted to improve the overview in a directory. By default, these are the `.aux` and `.log` files: `cleanup={aux,log}`.

`framesep` Value for `\fbox` if keyword `frame` is used.

`mpsep` Distance between code and output (default 1 em).

`pagesep` Distance between pages for multipage output (default 1 em).

`shiftFN` Length to shift marginal filename; positive values shift up.

`inline` False by default; if true, include the generated output in the paragraph, not as a display.

`showoutput` True by default; if false, omit the generated output.

`code` True by default (unless `inline=true`); if false, omit the source code listing.

`tcbox` If `false`, do not use any box commands from the `tcolorbox` package (for debugging and in case of bugs).

`eps` Convert the generated PDF file to EPS (historical reasons).

10 Caveats

Due to issues with `tcolorbox`, you can expect problems if a page break appears in the code part immediately before the first code line is printed. In such a case put a `\newpage` or `\pagebreak` just *before* the `externalDocument` environment. If you do not need `tcolorbox` features, you can disable its use with the option `tcbox=false`.

References

[1] C. Heinz, J. Hoffmann, B. Moses. The listings package, version 1.8d, 2020-03-24. `ctan.org/pkg/listings`

[2] R. Niepraschk. The showexpl package, version 0.3s. `ctan.org/pkg/showexpl`

[3] T.F. Sturm. The tcolorbox package, version 5.0.2, 2022-01-07. `ctan.org/pkg/tcolorbox`

[4] T. Van Zandt, H. Voß, et al. The fancyvrb package, version 4.2. `ctan.org/pkg/fancyvrb`

[5] H. Voß. The hvextern package, version 0.28. `ctan.org/pkg/hvextern`

◇ Herbert Voß
Wasgenstraße 21
14129 Berlin, Germany
`herbert (at) dante dot de`
`https://hvoss.org/`

The `luamodulartables` and `luaset` LaTeX packages

Chetan Shirore, Ajit Kumar

Abstract

The `luamodulartables` package was developed by us to generate modular addition and multiplication tables for positive integers, for use in LaTeX documents. The commands in the package have optional arguments for formatting of tables. These commands can be used in an environment similar to the `tabular` and `array` environments. The commands can also be used with the `booktabs` package to provide better formatting of tables in LaTeX.

Similarly, the `luaset` package is developed by us to define finite sets and perform different operations on them inside LaTeX documents. There is no special environment in the package for performing set operations. The package commands can be used in any environment (including mathematics environment).

These packages are written in Lua, and the TeX source is to be compiled with the LuaLaTeX engine. There is no need to install Lua on users' systems as TeX distributions (TeX Live and MiKTeX) come bundled with LuaLaTeX. The packages can be modified or extended by writing custom Lua programs.

1 Introduction

The Lua [1] programming language is a scripting language which can be embedded across platforms. With LuaTeX [3], and more easily with the `luacode` [2] package, it is possible to use Lua in LaTeX. The TeX [9] and LaTeX languages provide for programming [8]. However, with the internals of TeX there are several limitations, especially for performing calculations on numbers in LaTeX documents. There are packages like `pgf` [7] and `xparse` [10] in LaTeX which provides some programming capabilities inside LaTeX documents. However, such packages do not provide the complete programming structure that general programming languages, such as Lua, provide. The `luacode` [2] package is used in development, in addition to the `xkeyval` package.

The modular addition (multiplication) of integers with respect to a positive integer n is obtained by taking the remainder of the usual addition (multiplication) after dividing it by n. There is no easy way in LaTeX to do modular addition and multiplication [4]. With Lua, it can be achieved easily in LaTeX. Also, non-Lua ways of doing modular arithmetic in LaTeX are more complicated [5].

The time required for the LuaTeX compilation to generate modulo addition and multiplication tables with the `luamodulartables` package, or to perform different operations on sets with the `luaset` package, is not an issue.

2 Installation and license

The installation of `luamodulartables` and `luaset` package is similar to simple LaTeX packages, with a `.sty` file in the LaTeX directory of a `texmf` tree. The packages can be loaded with `\usepackage{luaset}` and `\usepackage{luamodulartables}` commands in the preamble of a LaTeX document. The TeX file is to be compiled using the LuaLaTeX engine.

`luamodulartables` and `luaset` packages are released under the LaTeX Project Public License v1.3c or later. The complete license text is available at `latex-project.org/lppl.txt`. The packages are developed in Lua. Lua is available as certified open source software. Its license is simple and liberal, compatible with the GNU GPL. A small part of the development of these packages was inspired by questions on `https://tex.stackexchange.com`. The content on this site is available under the CC BY-SA license.

3 The `luamodulartables` package

`\luaModularMult` and `\luaModularAdd` are the two basic commands in the `luamodulartables` package, to generate modular multiplication and addition tables, respectively. The command `\luaModularMult` has the following syntax and it is used to generate modular multiplication tables for positive integers.

```
\luaModularMult
  [multlabel=⟨text⟩,
   headline=⟨text⟩,midline=⟨text⟩]
  {⟨n⟩}
```

The command has one compulsory argument $\langle n\rangle$, and three optional arguments `multlabel`, `headline` and `midline`. The compulsory argument denotes the positive integer n with respect to which modular multiplication is to be carried out.

The `multlabel` string denotes the label to be printed as the entry in the first row and first column of the generated `tabular` environment. Its default value is `$\times$`. The `headline` refers to the style of horizontal line after first row in tabular or table environment. The `midline` refers to the style of horizontal lines after second row till the second last row. The `headline` and `midline` strings are both empty by default.

The formatting of the top line (before the beginning of the first row) and the bottom line (after the end of the last row) are defined in the user's LaTeX document. The alignment of columns and use of vertical lines for columns are likewise specified in the document.

doi.org/10.47397/tb/43-3/tb135shirore-luamath

$\mathbb{Z}_4$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Table 1: Illustration of `\luaModularMult`

An example of using the `\luaModularMult` command follows, specifying the optional arguments `multlabel` and `headline`. It requires the `amsmath` and `amssymb` packages.

```
\begin{tabular}{r|rrrr} \toprule
\luaModularMult[multlabel=$\mathbb{Z}_4$,
                headline=\midrule]
               {4} \\
\bottomrule \end{tabular}
```

This generates the output shown in Table 1.

Similarly, the command `\luaModularAdd` is used to generate addition modulo tables for positive integers. It has the following syntax:

```
\luaModularMult
  [addlabel=⟨text⟩,
   headline=⟨text⟩,midline=⟨text⟩]
  {⟨n⟩}
```

The `addlabel` argument denotes the label to be printed as the entry in the first row and the first column of tabular environment. Its default value is `$+$`. The optional parameters `headline` and `midline` are exactly the same as in the `\luaModularMult` command.

4 The `luaset` package

4.1 Defining and displaying sets

A set can be defined with the `\luaSetNew` command:

```
\luaSetNew{⟨name⟩}{⟨set⟩}
```

For example, the following defines sets A and B:

```
\luaSetNew{A}{a,b,c,10,d,10,a,30}
\luaSetNew{B}{d,e,f,10,20}
```

The set can be output with `\luaSetPrint`:

```
\luaSetPrint{⟨name of set⟩}
```

Continuing our example, the commands

```
\(A = \luaSetPrint{A}\) \\
\(B = \luaSetPrint{B}\)
```

generates this output (notice that duplicate elements have been removed, this being a set):

$A = \{10, 30, a, b, c, d\}$
$B = \{10, 20, d, e, f\}$

The command `\luaSetPrint` parses, sorts and prints all elements by using the `parsers.iterator` [6] function in `lualibs`.

4.2 Commands in the `luaset` package

These commands are available in the `luaset` package to perform various operations on finite sets in LaTeX documents.

`\luaSetUnion{C}{A}{B}` Defines new set C as union of sets A and B.

`\luaSetIntersection{C}{A}{B}` Defines new set C as intersection of sets A and B.

`\luaSetDifference{C}{A}{B}` Defines new set C as difference of sets A and B.

`\luaSetCardinal{A}` Gives cardinality of set A.

`\luaSetBelongsto{x}{A}` Returns true if x is in set A, otherwise returns false.

`\luaSetSubseteq{A}{B}` Returns true if set A is a subset of set B, otherwise returns false.

`\luaSetSubset{A}{B}` Returns true if set A is a proper subset of set B, otherwise returns false.

`\luaSetEqual{A}{B}` Returns true if set A is equal to set B, otherwise returns false.

References

[1] Lua 5.4 reference manual. lua.org/manual/5.4
[2] Luacode package page. ctan.org/pkg/luacode
[3] LuaTeX package page. ctan.org/pkg/luatex
[4] Modular arithmetic in LaTeX. mathoverflow.net/questions/18813
[5] Modular arithmetic in LaTeX. tex.stackexchange.com/questions/34424
[6] parsers.iterator function in lualibs. github.com/latex3/lualibs/blob/main/lualibs-util-prs.lua
[7] PGF package page. ctan.org/pkg/pgf
[8] W.M. Richter. TeX and scripting languages. *TUGboat* 25(1):71–88, 2004. tug.org/TUGboat/tb25-1/richter.pdf
[9] Wikipedia. TeX. en.wikipedia.org/wiki/TeX
[10] Xparse package page. ctan.org/pkg/xparse

◇ Chetan Shirore
Department of Mathematics,
Institute of Chemical
Technology, Mumbai - 400019,
Maharashtra, India

◇ Ajit Kumar
Department of Mathematics,
Institute of Chemical
Technology, Mumbai - 400019,
Maharashtra, India

Using OpenType and TrueType fonts with XƎLATEX and LuaLATEX

Herbert Voß

Abstract

For both "new" TEX engines XƎTEX and LuaTEX, which are admittedly no longer all that new, there are a few things to consider in connection with fonts that are important for users who have previously worked with pdfLATEX. Until now, only the fonts that came with the TEX distribution in use were easily available to "inexperienced" users.

1 Introduction

With the use of XƎLATEX or LuaLATEX the following facts have to be considered:

- The default font is by definition Latin Modern, regardless of whether `fontspec` is loaded or not. For pdfLATEX, the default is Computer Modern.
- The packages `inputenc` and/or `fontenc` should not be loaded. First, UTF-8 is already for years the all-but-universal default input encoding, and second, `fontspec` automatically loads `fontenc` with the TU (Unicode) encoding.
- To support OpenType fonts for math, load the `unicode-math` package instead of the `amsfonts` and/or `amssymb` packages. The `amsmath` package can still be used, but should be loaded *before* `fontspec`/`unicode-math`.
- The `xltxtra` and `xunicode` packages are obsolete and should no longer be used.

The `fontspec` package can be used with both XƎLATEX and LuaLATEX:

`\usepackage[`⟨*options*⟩`]{fontspec}`

It greatly simplifies the integration of OpenType and TrueType fonts that are not part of the TEX distribution. In order for the system to find them automatically, they must either be in the current document directory or, depending on the operating system, in (usually) one of the following directories:

GNU/Linux	`/usr/share/fonts`
	`/usr/local/share/fonts`
	`~/.fonts` (user-specific)
Windows	`C:\Windows\Fonts`
macOS	`/System/Library/Fonts`
	`/Library/Fonts`
	`~/Library/Fonts` (user-specific)

Any other readable directory can be used, if the path is passed to the package `fontspec`.

First published in *Die TEXnische Komödie* 1/2022, pp. 49–57; translation by the author.

The properties of a font can be displayed with an appropriate program, e.g. `otfinfo`, which is available on every TEX distribution. For a font which is part of the TEX directory tree, one can simplify the argument: the complete path is not needed, since it can be found with `kpsewhich`:

```
$ otfinfo -i ‘kpsewhich ComicNeue-Regular.otf‘
Family:             Comic Neue
Subfamily:          Regular
Full name:          Comic Neue Regular
PostScript name:    ComicNeue-Regular
Version:            Version 2.003;hotconv 1.0.109
Unique ID:          2.003;;Comic Neue Light
Designer:           Craig Rozynski
Designer URL:       http://www.craigrozynski.com
Manufacturer:       Craig Rozynski
Vendor URL:         http://www.comicneue.com
Copyright:          Copyright 2014 The Comic Neue
    Project Authors (https://github.com/crozynski/
    comicneue)
License URL:        https://scripts.sil.org/OFL
License Description: This Font Software is licensed
    under the SIL Open Font License, Version 1.1. [...]
Vendor ID:          UKWN
```

2 Font search: `luaotfload-tool` and `luafindfont`

The program `luaotfload-tool` can be used to list the fonts installed on the system. Both system fonts and TEX distribution fonts are taken into account. However, the program is geared more to the needs of TEX itself than to those of users. For example, searching for the font "Times" typically returns something like this:

```
$ luaotfload-tool --find=times
luaotfload | resolve : Font "times" found!
luaotfload | resolve : Resolved file name "/System/
    Library/Fonts/Times.ttc", subfont nr. 0
```

That is, the output of the script is only one font, although several variants are installed, albeit with different file names. The search can be expanded by using the `--fuzzy` option. Now searching for "times new roman", for example:

```
$ luaotfload-tool --fuzzy --find="times new roman"
luaotfload | resolve : Font "times new roman" found!
luaotfload | resolve : Resolved file name "/System/
    Library/Fonts/Supplemental/Times New Roman.ttf"
```

But even this search is not especially successful. With the Lua utility `luafindfont`, which is part of the TEX distribution, searching for *times* yields more informative results (see Listing 1).

The `luafindfont` program can be started with various options and also allows an AND condition when specifying the font to search for. A brief listing of the options:

```
-h,--help
-i,--info     font number to use (default 0)
```

doi.org/10.47397/tb/43-3/tb135voss-unifont

```
-m,--max_string (default 90)
-n,--no-symbolic-names
-o,--otfinfo  font number and options (default 0)
-v,--verbose  verbose output
-x,--xetex
⟨string⟩
```

The main ⟨*string*⟩ argument is usually (part of) a font name, but can contain extra conditions, as explained below. A longer description of the options:

- **-i** ⟨***number***⟩ The existing font styles are output for the font with the specified number.
- **-m** ⟨***number***⟩ Number of characters to be used for the output of the font name including the path. The full path specification can be very long and thus can be limited, for example, to 50 characters by specifying `-m 50`. Characters in the middle of the path are replaced with "`...`".
- **-n** Omit the symbolic (family) names column from the output.
- **-o** ⟨***number***[⟨***option***⟩]⟩ The `otfinfo` program is run on the font with the specified number to supply additional font information. Options for `otfinfo` must immediately follow the font number.
- **-x** Test if font is found by `kpsewhich` (1/0).

Some example applications of `luafindfont`:

```
# search for times in the filename with path:
luafindfont times

# search for palatino and run otfinfo -v on font no 3:
luafindfont palatino -o 3v

# search for arial, font no 3, max 50 characters:
luafindfont -i 3 -m 50 arial
```

To search for both a font and style, the specifications are linked with the '&' character, which must be quoted for the shell. As an example, Listing 2 searches for all the Myriad semibold fonts.

Various options are supported by the `otfinfo` program, with its `i` option being the default. Listing 3 has an example of its output.

```
otfinfo options
  a   Report font's family name.
  f   Report font's GSUB/GPOS features.
  g   Report font's glyph names.
  i   Report font's names and designer/vendor info.
  p   Report font's PostScript name.
  s   Report font's supported scripts.
  t   Report font's OpenType tables.
  v   Report font's version information.
  z   Report font's optical size information.
```

The following examples are marked with a filename in the margin. All example files can be downloaded from `https://tug.org/~hvoss/tb135.zip`. These are complete documents whereas in this article I show only the important parts of the preamble and document body, with the output in a frame below. All examples must be run with LuaLaTeX.

3 Font selection by name

Selecting a font by its symbolic name assumes the font can be found in one of the directories listed above. XeLaTeX and LuaLaTeX go different ways: XeTeX searches for its fonts with `fontconfig`; this library (`freedesktop.org/www/Software/fontconfig`) is included in XeTeX, while LuaTeX determines the fonts from a self-created font catalogue. However, the normal user does not have to be particularly interested in this.

`tb135voss-1.tex`

```
\usepackage{fontspec}
```

```
\fontspec{Cambria}A short test with the font Cambria
and now a change to {\fontspec{DejaVu Serif}%
[Scale=0.85] the font Dejavu and now at last
{\fontspec{Arial}[Scale=0.9]to the font Arial.}}
```

A short test with the font Cambria and now a change to the font Dejavu and now at last to the font Arial.

As shown above, the correct font names can be found using various programs, such as `pdffonts`. Listing 4 shows the fonts used in the above example.

4 Font selection by file name

This variant should always be used for XeTeX if you want to use OpenType or TrueType fonts from the existing TeX tree, which are normally not recognized by the underlying operating system. This restriction does not exist for LuaTeX; TeX font trees are also searched.

`tb135voss-2.tex`

```
\usepackage{fontspec}
```

```
\fontspec{Iwona}A test in the Iwona font and now a
switch to the {\fontspec{Kurier}Kurier font and now
to the {\fontspec{Antykwa Poltawskiego}Antykwa
Poltaws\-kiego font}} which doesn't look like
\fontspec{HelveticaNeue}Helvetica Neue.
```

A test in the Iwona font and now a switch to the Kurier font and now to the Antykwa Poltawskiego font which doesn't look like Helvetica Neue.

Since LuaTeX manages its own cache for the fonts, there is a pause in the TeX run the first time it is called because this cache has to be created:

```
[...]
luaotfload | db : Font names database not found,
    generating new one.
```

Listing 1: Searching for Times on my local system

```
$ luafindfont times
 No.                         Filename      Symbolic Name                              Path
   1.  Times New Roman Bold Italic.ttf      timesnewroman     /Users/voss/Library/Fonts/Times/
   2.  Times New Roman Bold Italic.ttf      timesnewroman  /System/Library/Fonts/Supplemental/
   3.         Times New Roman Bold.ttf      timesnewroman  /System/Library/Fonts/Supplemental/
   4.         Times New Roman Bold.ttf      timesnewroman     /Users/voss/Library/Fonts/Times/
   5.       Times New Roman Italic.ttf      timesnewroman     /Users/voss/Library/Fonts/Times/
   6.       Times New Roman Italic.ttf      timesnewroman  /System/Library/Fonts/Supplemental/
   7.              Times New Roman.ttf      timesnewroman  /System/Library/Fonts/Supplemental/
   8.              Times New Roman.ttf      timesnewroman     /Users/voss/Library/Fonts/Times/
   9.                        Times.ttc              times              /System/Library/Fonts/
  10.             Times_Sans_Serif.ttf     timessansserif     /Users/voss/Library/Fonts/Times/
  11.      TimesNewRomanMTStd-Bold.otf timesnewromanmtstd     /Users/voss/Library/Fonts/Times/
  12.  TimesNewRomanMTStd-BoldCond.otf timesnewromanmtstd     /Users/voss/Library/Fonts/Times/
  13.    TimesNewRomanMTStd-BoldIt.otf timesnewromanmtstd     /Users/voss/Library/Fonts/Times/
  14.      TimesNewRomanMTStd-Cond.otf timesnewromanmtstd     /Users/voss/Library/Fonts/Times/
  15.    TimesNewRomanMTStd-CondIt.otf timesnewromanmtstd     /Users/voss/Library/Fonts/Times/
  16.    TimesNewRomanMTStd-Italic.otf timesnewromanmtstd     /Users/voss/Library/Fonts/Times/
  17.           TimesNewRomanMTStd.otf timesnewromanmtstd     /Users/voss/Library/Fonts/Times/
  18. TimesNewRomanPS-BoldItalicMT.otf  timesnewromanpsmt     /Users/voss/Library/Fonts/Times/
  19.       TimesNewRomanPS-BoldMT.otf  timesnewromanpsmt     /Users/voss/Library/Fonts/Times/
  20.            TimesNewRomanPSMT.otf  timesnewromanpsmt     /Users/voss/Library/Fonts/Times/
  21.      TimesNewRomanPSStd-Bold.otf timesnewromanpsstd     /Users/voss/Library/Fonts/Times/
  22.    TimesNewRomanPSStd-BoldIt.otf timesnewromanpsstd     /Users/voss/Library/Fonts/Times/
  23.    TimesNewRomanPSStd-Italic.otf timesnewromanpsstd     /Users/voss/Library/Fonts/Times/
  24.   TimesNewRomanPSStd-Regular.otf timesnewromanpsstd     /Users/voss/Library/Fonts/Times/
```

```
luaotfload | db : This can take several minutes; please
     be patient.(compiling luc: /home/voss/texlive/2022/
     texmf-var/luatex-cache/generic/fonts/otf/lmroman10-
     regular.luc)(compiling luc: /home/voss/.texlive2022
     /texmf-var/luatex-cache/generic/fonts/otf/lmroman10
     -regular.luc)(save: /home/voss/texlive/2022/texmf-
     var/luatex-cach
[...]
```

Running the example above with XƎLaTeX instead of LuaLaTeX will produce an error message because the fonts Iwona, Kurier, and Antykwa Poltawskiego will not be found by XƎTeX, since a full file name is not specified. In the following example, the file name with an extension must therefore be specified for the first three fonts, whereas `HelveticaNeue` is still loaded via the name, since it is a system font in my local macOS system and not part of the TeX distribution.

tb135voss-3.tex

```
\usepackage{fontspec}
```

```
\fontspec{Iwona-Regular.otf} A test in the font Iwona
and now a change to the font
{\fontspec{Kurier-Regular.otf} Kurier and now to
{\fontspec{antpolt-regular.otf} most recently the
font Antykwa Poltawskiego,}} which doesn’t look like
\fontspec{HelveticaNeue} Helvetica.
```

A test in the font Iwona and now a change to the font Kurier and now to most recently the font Antykwa Poltawskiego, which doesn't look like Helvetica.

The file extension can be specified using the optional argument `Extension` and a directory that is not in the normal search path using the `Path` option. In this case, however, only the specified font style is activated; in the following example, `\bfseries` does not display bold because no corresponding bold variant was declared or was not found by `fontspec`.

tb135voss-4.tex

```
\usepackage{fontspec}
\setmainfont{BertholdWalbaumBook.ttf}
```

```
A test with the Berthold Walbaum font:\par
A completely normal text in the old beautiful
font, which was \bfseries embedded as TrueType.
```

A test with the Berthold Walbaum font:
A completely normal text in the old beautiful font, which was embedded as TrueType.

The bold variant can be assigned using the optional argument `BoldFont`:

Listing 2: Searching for a font with a special shape

```
$ luafindfont -i 4 "myriad & semibold"
 No.                       Filename Symbolic Name                             Path
   1.          MyriadPro-Semibold.otf myriadpro  /Users/voss/Library/Fonts/MyriadPro/
   2.      MyriadPro-SemiboldCond.otf myriadpro  /Users/voss/Library/Fonts/MyriadPro/
   3.    MyriadPro-SemiboldCondIt.otf myriadpro  /Users/voss/Library/Fonts/MyriadPro/
   4.        MyriadPro-SemiboldIt.otf myriadpro  /Users/voss/Library/Fonts/MyriadPro/
   5.    MyriadPro-SemiboldSemiCn.otf myriadpro  /Users/voss/Library/Fonts/MyriadPro/
   6.  MyriadPro-SemiboldSemiCnIt.otf myriadpro  /Users/voss/Library/Fonts/MyriadPro/
   7.   MyriadPro-SemiboldSemiExt.otf myriadpro  /Users/voss/Library/Fonts/MyriadPro/
   8. MyriadPro-SemiboldSemiExtIt.otf myriadpro  /Users/voss/Library/Fonts/MyriadPro/

Font: myriadpro
Fonttype otf(system) --> | Regular | Bold | Italic | BoldItalic |
```

Listing 3: Printing OpenType features

```
$ luafindfont -o 2f "myriad & semibold"
(output from Listing 2 omitted)
Running otfinfo -f for font no.2
otfinfo -f "/Users/voss/Library/Fonts/MyriadPro/
     MyriadPro-SemiboldCond.otf"
aalt    Access All Alternates
case    Case-Sensitive Forms
cpsp    Capital Spacing
dnom    Denominators
fina    Terminal Forms
frac    Fractions
kern    Kerning
liga    Standard Ligatures
lnum    Lining Figures
numr    Numerators
onum    Oldstyle Figures
ordn    Ordinals
pnum    Proportional Figures
sinf    Scientific Inferiors
sups    Superscript
tnum    Tabular Figures
zero    Slashed Zero
```

tb135voss-5.tex

```
\usepackage{fontspec}
\setmainfont{BertholdWalbaumBook}
  [Path=fonts/,
   Extension=.ttf,
   BoldFont=BertholdWalbaumMediumBook]
```

```
A test with the Berthold Walbaum font:

A completely normal text in the beautiful old font,
which was \bfseries integrated as TrueType.
```

A test with the Berthold Walbaum font:
A completely normal text in the beautiful old font, which was **integrated as TrueType.**

Entering the fonts via a name can be simplified when using LuaLaTeX if you work with the wildcard ∗. Then the part of the name that is the same for all variants needs to be specified only for the base name.

In the following example, the fonts are only found using the base name BertholdImagoBQ, whereby this base name itself is not a font name. Therefore, a definition must also be made for `UprightFont`.

```
BertholdImagoBQ-Book.otf
BertholdImagoBQ-BookItalic.otf
BertholdImagoBQ-MediumItalic.otf
BertholdImagoBQ-Medium.otf
```

tb135voss-6.tex

```
\usepackage{fontspec}
\setmainfont{BertholdImagoBQ}[
  UprightFont=*-Book,
  ItalicFont=*-BookItalic,
  BoldItalicFont=*-MediumItalic,
  BoldFont=*-Medium]
```

```
A test with the Berthold Imago font:\par
A completely normal text in the beautiful
new font, which was integrated as
\textbf{OpenType}. \textit{The font as
Italic and now additionally
\bfseries as a bold variant}
```

A test with the Berthold Imago font:
A completely normal text in the beautiful new font, which was integrated as **OpenType**. *The font as Italic and now additionally* ***as a bold variant***

5 Font families

With the previous examples, the main font was defined in each case, which can also be changed later by further calls; thus `\setromanfont` is rarely used, since it generally corresponds to the main font. The old syntax with `\setmainfont[`⟨*options*⟩`]{`⟨*font name*⟩`}` is still possible for all macros for reasons of compatibility.

In general, defining the fonts and their associated features is very time-consuming if the naming of the fonts is not organized in such a way that the `fontspec` package can do the assignment itself.

Listing 4: Font list of the first example pdf

```
$ pdffonts tb135voss-1.pdf
name                                 type              encoding         emb sub uni object ID
------------------------------------ ----------------- ---------------- --- --- --- ---------
EKHFKG+Cambria                       CID Type 0C       Identity-H       yes yes yes      8  0
DSBAVG+DejaVuSerif                   CID TrueType      Identity-H       yes yes yes      9  0
CELWVW+ArialMT                       CID TrueType      Identity-H       yes yes yes     10  0
```

There are many packages that relieve the user of this work. As of this writing, CTAN lists 61 packages which do all the font setting internally. Here are some of the more commonly-used ones (capitalized according to the `.sty` filename, as distributed): `accanthis, Alegreya, bitter, cantarell, CharisSIL, Chivo, CormorantGaramond, crimson, CrimsonPro, dejavu-otf, droidsans, droidserif, ebgaramond, garamondlibre, gfsneohellenicot, imfellEnglish, kpfonts-otf, lato, lexend, libertinus-otf, librebaskerville, LibreBodoni, librecaslon, linguisticspro, marathi, newpxtext, newtxtext, noto, noto-serif, opensans, plex-otf, plex-serif, quattrocento, roboto, Rosario, sourceserifpro`

The complete list, with links, is available at `https://hvoss.org/Books/fontpackages.html`. For more information about any package, you can visit `https://ctan.org/pkg/`⟨*name*⟩.

We already differentiate between the TeX engines used and independently load the required font formats. Also, all font packages also load `fontspec` by default. As a final example, we show the main part of the package file `Alegreya.sty`:

Font package

```
[...]
\ifAlegreya@otf
  \RequirePackage{fontspec}
\else
  \RequirePackage{fontenc,fontaxes,mweights}
\fi

\ifAlegreya@otf
  \setmainfont[
    Numbers        = {\Alegreya@figurealign,
                      \Alegreya@figurestyle},
    UprightFont    = *-\Alegreya@regstyle ,
    ItalicFont     = *-Italic,
    BoldFont       = *-\Alegreya@boldstyle ,
    BoldItalicFont = *-\Alegreya@boldstyle Italic ,
  ]{Alegreya}
[...]
```

By virtue of this work in the package, the user need only write the `\usepackage` command shown below. In contrast, we use another font, Anonymous-Pro, for the typewriter text, for which there exists no font package and thus we have to use `\setmonofont` explicitly.

tb135voss-7.tex

```
\usepackage{Alegreya,AlegreyaSans}
\setmonofont[FakeStretch=0.8,
  Scale=MatchLowercase]{AnonymousPro}
```

```
The normal font is \textsc{Alegreya}, which is
also possible as \textbf{bold}.\par
\sffamily The sans serif is \textsc{Alegreya Sans},
which is also available
in \textbf{bold}.\par
\ttfamily And the mono font is Anonymous Pro, which
is yet again available as a \textbf{bold font}.\par
\addfontfeature{FakeStretch=0.65}We
can further condense the mono font;
the \textbf{bold version} gets the same treatment.
```

The normal font is ALEGREYA, which is also possible as **bold**.

The sans serif is ALEGREYA SANS, which is also available in **bold**.

`And the mono font is Anonymous Pro, which is yet again available as a` **`bold font`**.

`We can further condense the mono font; the` **`bold version`** `gets the same treatment.`

◇ Herbert Voß
Wasgenstraße 21
14129 Berlin, Germany
`herbert (at) dante dot de`
`https://hvoss.org/`

New directions in math fonts

Hans Hagen, Mikael P. Sundqvist

1 Introduction

After trying to improve math rendering of OpenType math fonts, the authors have ended up with a mix of improving the engine and fixing fonts runtime, and we are rather satisfied with the results so far.

However, as we progress and also improve the more structural and input-related features of ConTEXt, we wonder why we aren't more drastic when it comes to fonts. The OpenType specifications are vague, and most existing OpenType math fonts use a mixture of the OpenType features and the old TEX habits, so we are sort of on our own. The advantage of this situation is that we feel free to experiment and do as we like.

In another article we discuss our issues with Unicode math, and we have realized that good working solutions will be bound to a macro package anyway. Also, math typesetting has not evolved much after Don Knuth set the standard, even if the limitations of those times in terms of memory, processing speed and font technologies have been lifted for quite a while. And right from the start Don invited users to extend and adapt TEX to one's needs.

Here we will zoom in on a few aspects: font parameters, glyph dimensions and properties and kerning of scripts and atoms. We discuss OpenType math fonts only, and start with a summary of how we tweak them. We leave a detailed engine discussion to a future article, since that would demand way more pages, and could confuse the reader.

2 Tweaks, also known as goodies

The easiest tweaks to describe are those that wipe features. Because the TEX Gyre fonts have many bad top accent anchors (that is, the anchors sit above the highest point of the shape) the `wipeanchors` tweak can remove them, and we do that per specified alphabet.

$$\hat{7}$$

In a similar fashion we `wipeitalics` (italic corrections) from upright shapes. Okay, maybe they can play a role for subscript placement, but then they can also interfere, and they do not fit with the OpenType specification. The `wipecues` tweak zeros the dimensions of the invisible times and friends so that they don't interfere, and `wipevariants` gets rid of bad variants of specified characters.

The fixers is another category, and the names indicate what gets fixed. Tweaks like these take lists of code points and specific properties to fix. We could leave it to your imagination what

`fixaccents` `fixanchors` `fixellipses` `fixoldschool` `fixprimes` `fixradicals` `fixslashes`

do, but here are some details.

Starting with `fixaccents`: Inconsistencies in the dimensions of accents make them jump all over the place so we normalize them. We support horizontal stretching at the engine level.

$$\widehat{a+b+c+d} = \widehat{u+v+w+x+y}$$

This required only a few lines of code, thanks to scaling features that were already present.

`fixanchors`: Anchors can be off so we fix these to look better, especially on italic shapes. We make sure that the automated sizing works consistently, as this is driven by width and overshoot.

`fixellipses`: Several kind of ellipses can be inconsistent with each other as well as with periods (both shape- and sizewise) so we deal with that.

`fixoldschool`: TEX (TFM) fonts have a limited set of widths, heights, and depths. We need to fix for instance fences of various size because we want to apply kerns to scripts on the four possible corners, for which we need to know the real height and depth,

`fixprimes`: Discussing primes would take many paragraphs so we stick to mentioning that they are a mess. We now have native prime support in the engine, and we assume properly dimensioned symbols to be used.

`fixradicals`: TFM dimensions for the parts of a radical (e.g., square root) sign are, shall we say, unusual. Let's fix them.

`fixslashes`: Slashes are used for skewed fractions so we'd better make sure they are set up right.

The `replacealphabets` tweak is a nice goodie of another kind. We use this to provide alternative script (roundhand) and calligraphic (chancery) alphabets (we have both natively in ConTEXt, although Unicode combines them in one alphabet). Many available OpenType math fonts come with one of the two alphabets only, some with roundhand and some with chancery. For the record: this tweak replaces the older `variants` tweak, which filtered scripts from a stylistic font feature.

We also use the `replacealphabets` tweak to drop in Arabic shapes so that we can do bidirectional math. In practice that doesn't truly boil down to a replacement but more to an addition. The `addmirrors` features accompanies this, and it is again a rather small extension to the engine to make sure we can do this efficiently: when a character is looked up we check a mirror variant when we are in r2l mode,

doi.org/10.47397/tb/43-3/tb135hagen-mathchange

just like we look up a smaller variant when we're in compact font mode (a ConTEXt feature).

$$\sum_{س=ب}^{ج} ى^{س} = ى^{ب} \frac{١ - ى^{ج-ب+١}}{١ - ى} \qquad (ى \neq ١)$$

Another application of `replacealphabets` is to drop in single characters from another font. We use this for instance to replace the 'not really an alpha' in Bonum by one of our own liking. Here we show the Bonum math italic 'a' and its original alpha, together with the modified alpha:

$$a + \alpha + \alpha$$

For this we ship a companion font. On our disks (and in the distribution) you can find, in the directory `/tex/texmf-fonts/fonts/data/cms/companion`:

```
RalphSmithsFormalScript-Companion.otf
TeXGyreBonumMath-Companion.otf
XITSMath-Companion.otf
```

These are efficient drop-ins that are injected by the `replacealphabets`, some under user control, some always. We tried to limit the overhead, and bidirectional math could be simplified, which also had the benefit that when one does tens of thousands of bodyfont switches a bit of runtime is gained.

There are more tweaks: `addactuarian` creates the relevant actuarial symbols which is a right-sided radical (the engine has support for two-sided radicals). It takes a bit of juggling with virtual glyphs and extensible recipes, but the results are rewarding.

$${}^{2}_{m|}\bar{A}^{1}_{x:\overline{n}|}$$

In a similar fashion we try to add missing extensible arrows with `addarrows`, bars with `addbars`, equals with `addequals` and again using the radical mechanism fourier notation symbols (like hats) with `addfourier`. That one involves subtle kerning because these symbols end up at the right top of a fence-like symbol.

$$\widehat{f * g * h}(\xi) = (f * g * h)\widehat{\ }(\xi)$$

This was one of the reasons to introduce a more advanced kerning mechanism in the engine, which is not entirely trivial because one has to carry around more information, since all this is font- and character-bound, and when wrapped in boxes that gets hard to analyze. The `addrules` tweak makes sure that we can do bars over and under constructs properly, and `addparts` is there to add extensible recipes to characters.

Some of these tweaks are not new and are also available in MkIV, but more as features (optionally driven by the goodie file). An example is `addscripts` that is there for specially positioned and scaled signs (high minus and such) but that tweak will probably be redone as part of "deal with all these plus and minus issues". The (dedicated to Alan Braslau) `addprivates` tweak is an example of this: we add specific variants for unary minus and plus that users can enable on demand, which in turn of course gives class-specific spacing, but we promised not to discuss those engine features here.

$$\int_1^2 \left[(x+2)^{\frac{1}{2}} - (x+2)^{-\frac{1}{2}} \right] dx$$

There is a handful of tweaks that deal with fixing glyph properties (in detail). We mention: `dimensions` and `accentdimensions` that can reposition in the bounding box, fix the width and italic correction, squeeze and expand, etc. The `kernpairs` tweak adds kern pairs to combinations of characters, while the `kerns` tweak provides a way to add top left, bottom left, top right and bottom right kerns — and those really make the results look better so we love it!

$$\left(\frac{1}{1+x^2}\right)^n \qquad x^2/(1+x)$$

The `margins` tweak sets margin fields that the engine can use to better calculate accent positioning over the base character. The same is true for `setovershoots` that can make accents lean over a bit. The `staircase` feature can be used to add the (somewhat complicated) OpenType kerns. From all this you can deduce that the engine has all types of kerning that OpenType requires, and more.

Accents as specified in fonts can be a pain to deal with, so we have more tweaks for them: `copyaccents` moves them to the right slots and `extendaccents` makes sure that we can extend them. Not all font makers have the same ideas about where these symbols should sit and what their dimensions should be.

The `checkspacing` tweak fixes bad or missing spacing related to Unicode character entries in the font, because after all, we might need them. We need to keep MathML in mind, for instance, which means: processing content that we don't see and that can contain whatever an editor puts in. The `replacements` feature replaces one character by another from the same font, while `substitutes` replaces a character by one from a stylistic feature.

Relatively late we added the `setoptions` feature which was needed to control the engine for specific fonts. The rendering is controlled by a bunch of options (think of kerning, italic correction, and such). Some are per font, many per class. Because we can (and do) use mixed math fonts in a document, we might need to adapt the engine-level options per font, and that is what this tweak does: it passes options to the font so that the engine can consult them and prefer them over the 'global' ones. We needed this for some fonts that have old school dimensions for extensibles (like Lucida), simply because they imitated Computer Modern. Normally that goes unnoticed, but, as mentioned before, it interferes with our optional kerning. The `fixoldschool` tweak sort of can fix that too so `setoptions` is seldom needed. Luckily, some font providers are willing to fix their fonts!

We set and configure all these tweaks in a so-called goodie file, basically a runtime module that returns a Lua table with specifications. In addition to the tweaks subtable in the math namespace, there is a subtable that overloads the font parameters: the ones that OpenType specifies, but also new ones that we added. In the next section we elaborate more on these font-bound parameters.

3 Font parameters

At some point in the upgrading of the math machinery we discussed some of the inconsistencies between the math constants of the XITS and STIX fonts. Now, one has to keep in mind that XITS was based on a first release of STIX that only had Type 1 fonts so what follows should not to be seen as criticism, but more as observations and reason for discussion, as well as a basis for decisions to be made.

One thing we have to mention in advance: we often wonder why weird and/or confusing stuff in math fonts goes unnoticed. We have some ideas:

- The user doesn't care that much how math comes out. This can easily be observed when you run into documents on the Internet or posts on forums. And publishers don't always seem to care either. Consistency with old documents sometimes seems to be more important than quality.
- The user switches to another math font when the current one doesn't handle its intended math domain well. We have seen that happen and it's the easiest way out when you have little control anyway (for instance when using online tools).
- The user eventually adds some skips and kerns to get things right, because after all TeX is also about tweaking.
- The user doesn't typeset math that is particularly complex. It's mostly inline math with an occasional alignment (also in text style) and very few multi-level displays (with left and right fences that span at most a fraction).

We do not claim to be perfect, but we care for details, so let's go on. Table 1 shows the math constants as they can be found in the STIX (two) and XITS (one) fonts. When you typeset with these fonts you will notice that XITS is somewhat smaller, so two additional columns show the values used to compensate for the axis height and accent base height. For the relevance column: (1) 'mandatory' means a design-related value the font designer must supply; (2) 'optional' means apparently redundant values that would normally be identical; (3) a blank cell (the vast majority) means a value likely needed to be configured at the macro/document level.

As you can see in the table, very few values are the same. So, what exactly do these constants tell us? You might even wonder why they are there at all. Just think of this: we want to typeset math, and we have an engine that we can control. We know how we want it to look. So, what do these constants actually contribute? Plenty relate to the height and depth of the nucleus and/or the axis. The fact that we have to fix some in the goodie files, and the further fact that we need more variables that control positioning, makes for a good argument to just ignore most of the ones provided by the font, especially when they seem somewhat arbitrary. Can it be that font designers are just gambling a bit, looking at another font, and starting from there?

The relationship between TeX's math font parameters and the OpenType math constants is not one-to-one. Mapping them onto each other is possible but font dependent. However, we can assume that the values of Computer Modern are leading.

The `AxisHeight`, `AccentBaseHeight` and `FlattenedAccentBaseHeight` are set to the x-height, a value that is defined in all fonts. The `SkewedFractionVerticalGap` also gets that value. Other variables relate to the em-width (or `\quad`), for instance the `SkewedFractionHorizontalGap` that gets half that value. Of course these last two then assume that the engine handles skewed fractions.

Variables that directly map onto each other are `StretchStackGapBelowMin` → `bigopspacing1`, `StretchStackTopShiftUp` → `bigopspacing3`, `StretchStackGapAboveMin` → `bigopspacing2`, `StretchStackBottomShiftDown` → `bigopspacing4`. However, these clash with other mappings: `UpperLimitGapMin` → `bigopspacing1`, `LowerLimitGapMin` → `bigopspacing2`, `UpperLimitBaselineRiseMin` → `bigopspacing3`, `LowerLimitBaselineDropMin` → `bigopspacing4`. While in traditional fonts these are the same, in OpenType they can be different. Should they be?

Internally we use different names for variables, simply because the engine has some parameters that

Table 1: OpenType math parameters, compared; bold indicates an unchanged value. See text for explanation of relevance.

constant	STIX	XITS	base	axis	relevance
`AccentBaseHeight`	450	480	**480**	464	optional**
`AxisHeight`	250	258	267	**258**	mandatory
`DelimitedSubFormulaMinHeight`	1500	1325	1600	1548	
`DisplayOperatorMinHeight`	1450	1800	1547	1496	
`FlattenedAccentBaseHeight`	662	656	706	683	optional**
`FractionDenominatorDisplayStyleGapMin`	198	150	211	204	
`FractionDenominatorDisplayStyleShiftDown`	700	640	747	722	
`FractionDenominatorGapMin`	66	68	70	**68**	
`FractionDenominatorShiftDown`	480	585	512	495	
`FractionNumeratorDisplayStyleGapMin`	198	150	211	204	
`FractionNumeratorDisplayStyleShiftUp`	580	640	619	599	
`FractionNumeratorGapMin`	66	68	70	**68**	
`FractionNumeratorShiftUp`	480	585	512	495	
`FractionRuleThickness`	66	68	70	68	optional
`LowerLimitBaselineDropMin`	600	670	640	619	
`LowerLimitGapMin`	150	135	160	155	
`MathLeading`	150	150	160	155	
`MinConnectorOverlap`	50	100	53	52	mandatory
`OverbarExtraAscender`	66	68	70	**68**	
`OverbarRuleThickness`	66	68	70	**68**	optional*
`OverbarVerticalGap`	198	175	211	204	
`RadicalDegreeBottomRaisePercent`	70	55	75	72	mandatory
`RadicalDisplayStyleVerticalGap`	186	170	198	192	
`RadicalExtraAscender`	66	78	70	68	
`RadicalKernAfterDegree`	−555	−335	-592	-573	
`RadicalKernBeforeDegree`	277	65	295	286	
`RadicalRuleThickness`	66	68	70	**68**	
`RadicalVerticalGap`	82	85	87	**85**	
`ScriptPercentScaleDown`	75	70	80	77	
`ScriptScriptPercentScaleDown`	60	55	64	62	
`SkewedFractionHorizontalGap`	300	350	320	310	
`SkewedFractionVerticalGap`	66	68	70	**68**	
`SpaceAfterScript`	41	40	44	42	
`StackBottomDisplayStyleShiftDown`	900	690	960	929	
`StackBottomShiftDown`	800	385	853	826	
`StackDisplayStyleGapMin`	462	300	493	477	
`StackGapMin`	198	150	211	204	
`StackTopDisplayStyleShiftUp`	580	780	619	599	
`StackTopShiftUp`	480	470	512	495	
`StretchStackBottomShiftDown`	600	590	640	619	
`StretchStackGapAboveMin`	150	68	160	155	
`StretchStackGapBelowMin`	150	68	160	155	
`StretchStackTopShiftUp`	300	800	320	310	
`SubSuperscriptGapMin`	264	150	282	272	
`SubscriptBaselineDropMin`	50	160	53	52	
`SubscriptShiftDown`	250	210	267	258	
`SubscriptTopMax`	400	368	427	413	
`SuperscriptBaselineDropMax`	375	230	400	387	
`SuperscriptBottomMaxWithSubscript`	400	380	427	413	
`SuperscriptBottomMin`	125	120	133	129	
`SuperscriptShiftUp`	400	360	427	413	
`SuperscriptShiftUpCramped`	275	252	293	284	
`UnderbarExtraDescender`	66	68	70	**68**	
`UnderbarRuleThickness`	66	68	70	**68**	optional*
`UnderbarVerticalGap`	198	175	211	204	
`UpperLimitBaselineRiseMin`	300	300	320	310	
`UpperLimitGapMin`	150	135	160	155	

OpenType math does not. So for `bigopspacing5`, we have `limit_above_kern` and `limit_below_kern`.

A couple of parameters have different values for (cramped) displaystyle:
`FractionDelimiterSize` → `delim2`,
`FractionDelimiterDisplayStyleSize` → `delim1`,
`FractionDenominatorShiftDown` → `denom2`,
`FractionDenominatorDisplayStyleShiftDown`
→ `denom1`, and their numerator counterparts from `num2` and `num1`. The `Stack*` parameters also use these. The `sub1`, `sub2`, `sup1`, `sup2`, `sup3`, `supdrop` parameters can populate the `Sub*` and `Super*` parameters, also in different styles.

The rest of the parameters can be defined in terms of the default rulethickness, quad or x-height, often multiplied by a factor. For some we see the `1/18` show up, a number we also see with muskips. Some constants can be set from registers, such as `SpaceAfterScript` which is just `\scriptspace`.

If you look at the LuaTEX source you will find a section where this mapping is done in the case of a traditional font, that is: one without a math constants table. In LuaMetaTEX we don't need to do this because font loading happens in Lua. So we simply issue an error when the math engine can't resolve a mandatory parameter. The fact that we have a partial mapping from math constants onto traditional parameters and that LuaTEX has to deal with the traditional ones too make for a somewhat confusing landscape. When in LuaMetaTEX we assume wide fonts to be used that have a math constants table, we can probably clean up some of this.

We need to keep in mind that Cambria was the starting point, and it did borrow some concepts from TEX. But TEX had parameters because there was not enough information in the glyphs! Also, Cambria was meant for Word, and a word processor is unlikely to provide the level of control that TEX offers, so it needs some directions with respect to e.g. spacing. Without user control, it has to come up with acceptable compromises. So actually the LuaMetaTEX math engine can be made a bit cleaner when we just get rid of these parameters.

So, which constants are actually essential? The `AxisHeight` is important and also design-related. By definition, this is where the minus sits above the baseline, and this is usually true even in practice. It is used for displacements of the baseline so that for instance fractions nicely align. When testing scripts anchored to fences we noticed that the parenthesis in XITS had too little depth while STIX had the expected amount. This relates to anchoring relative to the math axis.

Is there a reason why `UnderbarRuleThickness` and `OverbarRuleThickness` should differ? If not, then we only need a variable that somehow tells us what thickness fits best with the other top and bottom accents. It is quite likely the same as the `RadicalRuleThickness`, which is needed to extend the radical symbol. So, here three constants can be replaced by one design-related one. The parameter `FractionRuleThickness` can also be derived from that, but more likely is that it is a quantity that the macro package sets up anyway, maybe related to rules used elsewhere.

The parameters `MinConnectorOverlap` and `RadicalDegreeBottomRaisePercent` also relate to the design although one could abuse the top accent anchor for the second one. So they are important. However, given the small number of extensibles, they could have been part of the extensible recipes.

The parameters `AccentBaseHeight` and `FlattenedAccentBaseHeight` might relate to the margin that the designer put below the accent as part of the glyph, which is kind of a design-related constant. Nevertheless, we fix quite a lot of accents in the goodie files because they can be inconsistent. That makes these constants somewhat dubious too. If we have to check a font, we can just as well set up constants that we need in the goodie file. Also, isn't it weird that there are no bottom variants? (In OpenType; Knuth didn't need them for TAOCP.)

We can forget about `MathLeading` as it serves no purpose in TEX. The `DisplayOperatorMinHeight` is often set wrong so although we fix that in the goodie file it might be that we just can use an internal variable. It is not the font designer who decides that anyway. The same is true for the parameter `DelimitedSubFormulaMinHeight`.

If we handle skewed fractions, `SkewedFraction-HorizontalGap` and `SkewedFractionVerticalGap` might give an indication of the tilt but why do we need two? It is design-related though, so they have some importance, when set right.

The rest can be grouped, and basically we can replace them by a consistent set of engine parameters. We can still set them up per font, but at least we can then use a clean set. Currently, we already have more. For instance, why only `SpaceAfterScript` and not one for before, and how about prescripts and primes? If we have to complement them with additional ones and also fix them, we might as well set up all these script-related variables.

For fractions, the font provides:
`FractionDenominatorDisplayStyleGapMin`,
`FractionDenominatorDisplayStyleShiftDown`,
`FractionDenominatorGapMin`,

`FractionDenominatorShiftDown`, `FractionNumeratorDisplayStyleGapMin`, `FractionNumeratorDisplayStyleShiftUp`, `FractionNumeratorGapMin`, `FractionNumeratorShiftUp`. We might try to come up with a simpler model.

Limits have: `LowerLimitBaselineDropMin`, `LowerLimitGapMin`, `UpperLimitBaselineRiseMin`, `UpperLimitGapMin`. Limits are tricky anyway as they also depend on abusing the italic correction for anchoring.

Horizontal bars are driven by: `OverbarExtraAscender`, `OverbarVerticalGap`, `UnderbarExtraDescender`, `UnderbarVerticalGap`, but for e.g. arrows we are on our own, so again not such a useful set.

Then radicals; we need some more than these: `RadicalDisplayStyleVerticalGap`, `RadicalExtraAscender`, `RadicalKernAfterDegree`, `RadicalKernBeforeDegree`, `RadicalVerticalGap`. Because we definitely need to check and fix these, there is no gain having them in the font.

Isn't it more a decision by the macro package how script and scriptscript should be scaled? Currently we listen to `ScriptPercentScaleDown` and `ScriptScriptPercentScaleDown`, but maybe it relates more to usage.

We need more control than just `SpaceAfterScript` and an engine could provide it more consistently. It's a loner.

How about `StackBottomShiftDown`, `StackBottomDisplayStyleShiftDown`, `StackDisplayStyleGapMin`, `StackGapMin`, `StackTopDisplayStyleShiftUp`, `StackTopShiftUp`? And aren't these more for the renderer to decide: `StretchStackBottomShiftDown`, `StretchStackGapAboveMin`, `StretchStackGapBelowMin`, `StretchStackTopShiftUp`?

This messy bit can also be handled more conveniently, so what exactly is the relationship with the font design of: `SubSuperscriptGapMin`, `SubscriptBaselineDropMin`, `SubscriptShiftDown`, `SubscriptTopMax`, `SuperscriptBaselineDropMax`, `SuperscriptBottomMaxWithSubscript`, `SuperscriptBottomMin`, `SuperscriptShiftUp`, `SuperscriptShiftUpCramped`?

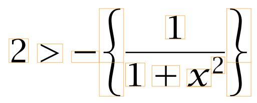
Just for the record, here are the (font-related) ones we added so far. A set of prime-related constants similar to the script ones: `PrimeBaselineDropMax`, `PrimeRaisePercent`, `PrimeRaiseComposedPercent`, `PrimeShiftUp`, `PrimeShiftUpCramped`, `PrimeSpaceAfter`, `PrimeWidthPercent`.

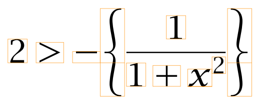
We also added `SpaceBeforeScript` just because we want to be symmetrical in the engine where we also have to deal with prescripts.

These we provide for further limit positioning: `NoLimitSupFactor`, `NoLimitSubFactor`; these for delimiters: `DelimiterPercent`, `DelimiterShortfall`; and these for radicals in order to compensate for sloping shapes: `RadicalKernAfterExtensible`, `RadicalKernBeforeExtensible` because we have double-sided radicals.

Finally, there are quite some (horrible) accent tuning parameters: `AccentBaseDepth`, `AccentBottomOvershoot`, `AccentBottomShiftDown`, `AccentExtendMargin`, `AccentFlattenedBaseDepth`, `AccentSuperscriptDrop`, `AccentSuperscriptPercent`, `AccentTopOvershoot`, `AccentTopShiftUp`, `FlattenedAccentBottomShiftDown`, `FlattenedAccentTopShiftUp`, but we tend to move some of that to the tweaks on a per accent basis.

Setting these parameters right is not trivial, and also a bit subjective. We might, for instance, assume that the math axis is set right, but alas, when we were fixing the less and greater symbols in Lucida Bright Math, we found that all symbols were designed for a math axis of 325, instead of the given value 313, and that difference can be seen. If you look closely, the points on the greater than sign and the braces are slightly below the minus sign in “Old Lucida” on the left, and aligned completely on the right. (The greater than sign is also larger in size. See the accompanying article on Lucida for more examples and discussion of this particular font.)


Old Lucida


New Lucida

The assumption is that the axis goes through the middle of the minus. Luckily it was relatively easy to fix these two symbols (they also had to be scaled, maybe they originate in the text font?) and adapt the axis. We still need to check all the other fonts, but it looks like they are okay, which is good because the math axis plays an important role in rendering math. It is one of the few parameters that has to be present and right. A nice side effect of this is that we end up discussing new (ConTEXt) features. One can for instance shift all non-character symbols down just a little and lower the math axis, to get a bit more tolerance in lines with many inline fractions, radicals or superscripts, that otherwise would result in interline skips.

A first step in getting out of this mess is to define *all* these parameters in the goodie file where we fix them anyway. That way we are at least not dependent on changes in the font. We are not a word processor so we have way more freedom to control matters. And preset font parameters sometimes do more harm than good. A side effect of a cleanup can be that we get rid of the evolved mix of uppercase and lowercase math control variables and can be more consistent. Ever since LuaTEX got support for OpenType, math constants' names have been mapped and matched to traditional TEX font parameters.

4 Metrics, especially italic corrections

By "metrics", we refer to the dimensions and other properties of math glyphs. The origin of digital math fonts is definitely Computer Modern and thereby the storage of properties is bound to the TFM file format. That format is binary and can be loaded fast. It can also be stored in the format, unless you're using LuaTEX or LuaMetaTEX where Lua is the storage format. A TFM file stores per character a width, height, depth and italic correction. The file also contains font parameters. In math fonts there are extensible recipes and there is information about next-in-size glyphs. The file has kerning and ligature tables too.

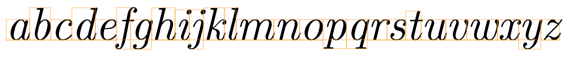
Given the times TEX evolved in, the format is rather compact. For instance, the height, depth and italic correction are shared and indices to three shared values are used. There can be only 16 distinct heights, 16 depths and 64 italic corrections. That way much fits into a memory word.

The documentation tells us that "The italic correction of a character has two different uses. (a) In ordinary text, the italic correction is added to the width only if the TEX user specifies '\/' after the character. (b) In math formulas, the italic correction is always added to the width, except with respect to the positioning of subscripts."

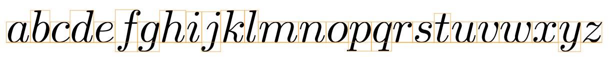
It is this last phenomenon that gives us some trouble with fonts in OpenType math. The fact that traditional fonts cheat with the width and that we add and selectively remove or ignore the correction makes for fuzzy code in LuaTEX, although splitting the code paths and providing options to control all this helps a bit. In LuaMetaTEX we have more control but also expect an OpenType font. In OpenType math there are italic corrections, and we even have the peculiar usage of it in positioning limits. However, the idea was that staircase kerns do the detailed relative positioning.

Before we dive into this a bit more, it is worth mentioning that Don Knuth paid a lot of attention to details. The italic alphabet in Computer Modern math uses nearly the same shapes as the CM text italic but metrics are quite different, as shown below. We have also met fonts where it looked like the text italics were used, and the math metrics handled via more excessive italic corrections, sometimes combined with staircase kerns that basically were corrections for the side bearing. This is why we always come back to Latin Modern and Cambria when we investigate fonts: one is based on the traditional TEX model, with carefully chosen italic corrections, and the other is based on the OpenType model with staircase kerning. They are our reference fonts.

Latin Modern Roman (text) italic:



Latin Modern Roman math italic:



In ConTEXt MkIV we played a lot with italic correction in math and there were ways to enforce, ignore, selectively apply it, etc. But, because fonts actually demand a mixture, in LuaMetaTEX we ended up with more extensive runtime patching of them. Another reason for this was that math fonts can have weird properties. It looks like when these standards are set and fonts are made, the font makers can do as they like as long as the average formula comes out right, and metrics to some extent resemble a traditional font. However, when testing how well a font behaves in a real situation there can be all kinds of interferences from the macro package: inter-atom kerning, spacing correction macros, specific handling of cases, etc. We even see OpenType fonts that seem to have the same limited number of heights, depths and italic corrections. And, as a consequence we get for instance larger sizes of fences having the same depth for all the size variants, something that is pretty odd for an OpenType font with no limitations.

The italic correction in traditional TEX math gets added to the width. When a subscript is attached to a kernel character it sits tight against that character: its position is driven by the width of the kernel. A superscript on the other hand is moved over the italic width so that it doesn't overlap or touch the (likely) "sticking out bit" of the kernel. This means that a traditional font (and many OpenType math fonts are modelled after Computer Modern) have to find compromises of width and italic correction for characters where the subscript is supposed to move left (inside the bounding box of the kernel).

The OpenType specification has some vague remarks about applying italic correction between the last in a series of slanted shapes and operators, as well as positioning limits, and suggests that it relates to relative super- and subscript positioning. It doesn't mention that the correction is to be added to the width. However, the main mechanism for anchoring scripts are these top and bottom edge kerns. This is why in fonts that provide these, we are unlikely to find italic correction unless it is used for positioning limits.

It is for that reason that an engine can produce reasonable results for fonts that either provide italics or provide kerns for anchoring: having both on the same glyph would mean troubles. It means that we can configure the engine options to add italic correction as well as kerns, assuming distinct usage of those features. For a font that uses both we need to make a choice (this is possible, since we can configure options per font). But that will certainly not lead to math that is always nicely typeset. In fact, without tweaks many fonts will still look right because in practice they use some mixture. But we are not aiming at partial success, we want all to look good.

Here is another thing to keep in mind (although now we are guessing a bit). There is a limited number of heights and depths in TEX fonts possible (16), but four times as many italic corrections can be defined (64). Is it because Don Knuth wanted to properly position the sub- and subscripts? Adding italic correction to the width is pretty safe: shapes should not overlap. Choosing the right width for a subscript needs more work because it's more visual. In the end we have a width that is mostly driven by superscript placement! That also means that as soon as we remove the italic correction things start looking bad. In fact, because upright math characters also have italic correction the term 'italic' is a bit of a cheat: it's all about script positioning and has little to do with the slope of the shapes.

4.1 Spacing

One of the reasons why for instance spacing between an italic shape and an upright one in TEX works out okay is that in most cases they come from a different font, which can be used as criterion for keeping the correction; between a sequence of same-font characters it gets removed. However, in OpenType math there is a good chance that all comes from the same font (at least in ConTEXt), unless one populates many families as in traditional TEX. We have no clue how other macro packages deal with this but it might well be the case that using many families (one for each alphabet) works better in the end. The engine is shape- and alphabet-agnostic, but one can wonder if we should add a glyph property indicating the distinctive range. It would provide engine level control over a run of glyphs (like multiplying a variable represented by a greek alpha by another variable represented by an upright b).

But glyph properties cannot be easily used here because we are still dealing with characters when the engine transforms the noad list into a node list. So, when we discussed this, we started wondering how the engine could know about a specific shape (and tilt) property at all, and that brought us to pondering about an additional axis of options. We already group characters in classes, but we can also group them with properties like `tilted`, `dotless`, `bold`. When we pair atoms we can apply options, spacing and such based on the specific class pair, and we can do something similar with category pairs.

It boils down to, for instance, a new `\mccode` that binds a character to a category. Then we add a command like `\setmathcategorization` (analogue to `\setmathspacing`) that binds options to pairs of categories. An easier variant of this might be to let the `\mccode` carry a (bit)set of options that then get added to the already existing options that can be bound to character noads as we create them. This saves us some configuration. Deciding what suits best depends on what we want to do: the fact that TEX doesn't do this means that probably no one ever gave it much thought, but once we do have this mechanism it might actually trigger demand, if only by staring at existing documents where characters of a different kind sit next to each other (take this 'a' invisible times 'x'). It would not be the first time that (in ConTEXt) the availability of some feature triggers creative (ab)usage.

4.2 Moving towards kerns

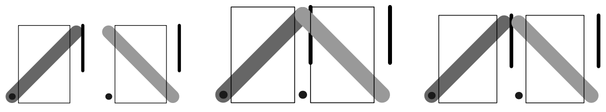
Because the landscape has settled, because we haven't seen much fundamental evolution in OpenType math, because in general TEX math doesn't particularly evolve, and because ConTEXt in the past has not been seen as suitable for math, we can, as mentioned before, basically decide what approach we follow. So, that is why we can pick up on this italic correction in a more drastic way: we can add the correction to the width, thereby creating a nicely bounded glyph, and moving the original correction to the right bottom kern, as that is something we already support. In fact, this feature is already available, we only had to add setting the right bottom kern. The good news is that we don't need to waste time on trying to get something extra in the font format, which is unlikely to happen anyway after two decades.

It is worth noticing that when we were exploring this as part of using MetaPost to analyze and visualize these aspects, we also reviewed the `wipeitalics` tweak and wondered if, in retrospect, it might be a dangerous one when applied to alphabets (for digits and blackboard bold letters it definitely makes sense): it can make traditional super- and subscript anchoring less optimal. However, for some fonts we found that improper bounding boxes can badly interfere anyway: for instance the upright 'f' in EB Garamond sticks out left and right, and has staircase kerns that make scripts overlap. The right top of the shape sticks out a lot and that is because the text font variant is used. We had already decided to add a `moveitalics` tweak that moves italic kerns into the width and then setting a right bottom kern that compensates it that can be a pretty good starting point for our further exploration of optimal kerns at the corners. That tweak also fixes the side bearings (negative llx) and compensates left kerns (when present) accordingly. An additional `simplifykerns` tweak can later migrate staircase kerns to simple kerns.

So, does all this free us from tweaks such as `dimensions` and `kerns`? Not completely. But we can forget about the italic correction in most cases. We have to set up fewer lower right kerns and maybe correct a few. It is just a more natural solution. So how about these kerns that we need to define? After all, we also have to deal with proper top kerns, and like to add kerns that are not there simply because the mentioned compromise between width, italic correction, and their combination was impossible. More about that in the next section.

5 Kerning

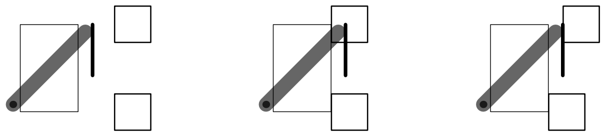
In the next pictures we will try to explain more visually what we have in mind and are experimenting with as we write this. In the traditional approach we have shapes that can communicate the width, height, depth and italic correction to the engine so that is what the engine can work with. The engine also has the challenge of anchoring subscripts and superscripts in a visually pleasing way.



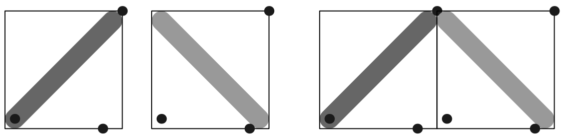
In this graphic we show two pseudo-characters. Each shown bounding box indicates the width as seen by the engine. An example of such a shape is the math italic 'f', and as it is used a lot in formulas it is also one of the hardest to handle when it comes to spacing: in nearly all fonts the right top sticks out and in some fonts the left part also does that. Imagine how that works out with scripts, fences and preceding characters.

When we put two such characters together they will overlap, and this is why we need to add the italic correction. That is also why the TEX documentation speaks in terms of "always add the italic correction to the width". This also means that we need to remove it occasionally, something that you will notice when you study for instance the LuaTEX source, that has a mix of traditional and OpenType code paths. Actually, compensating can be done either by changing the width property of a glyph node or by explicitly adding a kern. In LuaMetaTEX we always add real kerns because we can then trace better.

The last graphic in the above set shows how we compensate the width for the bit that sticks out. It also shows that we definitely need to take neighboring shapes into account when we determine the width and italic correction, especially when the latter is *not* applied (read: removed).

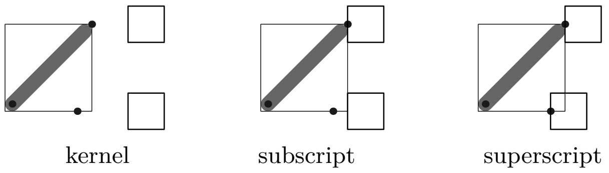


Here we anchored a super- and subscript. The subscript position is tight to the advance width, again indicated by the box. The superscript however is moved by the italic correction and in the engine additional spacing before and after can be applied as well, but we leave that for now. It will be clear that when the font designer chooses the width and italic correction, the fact that scripts get attached has to be taken into account.



In this graphic we combine the italic correction with the width. Keep in mind that in these examples we use tight values but in practice that correction can also add some extra right side bearing (white space). This addition is an operation that we can do when loading a font. At the same time we also compensate the left edge for which we can use the x-coordinate of the left corner of the glyph's real bounding box. The advance width starts at zero and that corner is then left of the origin. By looking at shapes we concluded that in most cases that shift is valid for usage in math where we don't need that visual overlap. In fact, when we tested some of that we found that the results can be quite horrible when you don't do that; not all fonts have left bottom kerning implemented.

The dot at the right indicates the old italic correction. Here we let it sit on the edge but as mentioned there can be additional (or maybe less) italic correction than tight.

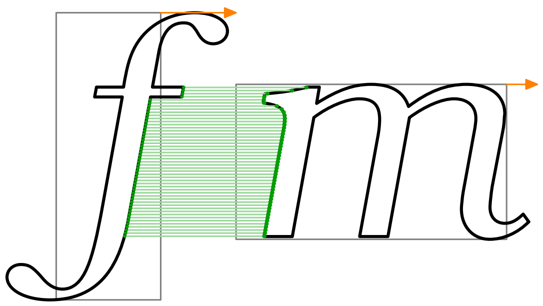


Finally we add the scripts here. This time we position the superscript and subscript at the top and bottom anchors. The bottom anchor is, as mentioned, the old italic correction, and the top one currently just the edge. And this is what our next project is about: identify the ideal anchors and use these instead.

In the ConTEXt goodie files (the files that tweak the math fonts at runtime) we can already set these top and bottom anchors and the engine will use them when set. These kerns are not to be confused with the more complicated staircase kerns. They are much simpler and lightweight. The fact that we already have them makes it relatively easy to experiment with this.

It must be noted that we talk about three kinds of kerns: inter-character kerns, corner kerns and staircase kerns. We can set them all up with tweaks but so far we've only done that for the most significant ones, like integrals. The question is: can we automate this? We should be careful because the bad top accent anchors in the TEX Gyre fonts demonstrate how flawed heuristics can be. It's interesting to remark that the developers of these font used MetaPost and are highly qualified in that area. And for us using MetaPost is also natural!

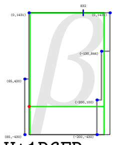
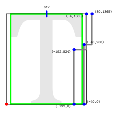
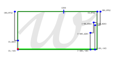
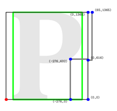
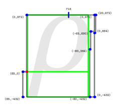
The approach that we follow is somewhat interactive. When working on the math update we like to chat (with Zoom) about these matters. We discuss and explore plenty and with these kerns we do the same. Because MetaPost produces such nice and crisp graphics, and because Metafun is well-integrated into ConTEXt we can link all these subsystems and just look at what we get. A lot is about visualization: if we discuss so-called 'grayness' as related to kerning, we end up with calculating areas, then look at what it tells us and as a next step figure out some heuristic. And of course we challenge each other into new trickery.



We are sure that getting this next stage in the perfection of math typesetting in ConTEXt and LuaMetaTEX will take quite some time, but the good news is that all machinery is in place. We also have to admit that it might not all work out well, so we end up sticking to what we have now. But at least we had the fun then. It is also a nice example of both applying mathematics and programming graphics.

That said, if it works out well, we can populate the goodie files with output from MetaPost, tweak a little when needed, and that saves us some time. One danger is that when we try to improve rendering the whole system also evolves which in turn will give different output, but we can always implement all this as features because after all ConTEXt is very much about configuration. And it makes for nice topics for articles and talks too!

The kerns discussed in the previous paragraphs are not the ones that we find in OpenType fonts. There we have 'staircase' kerns that stepwise go up or down by height and kern. So, one can have different kerns depending on the height and sort of follow the shape. This permits quite precise kerning between for instance the right bottom of a kernel and left top of a subscript. So how is that used in practice? The reference font Cambria has these kerns but close inspection shows that these are not that accurate. Fortunately, we never enter the danger zone with subscripts, because other parameters prevent that. If we look at for instance Lucida and Garamond, then we see that their kerns are mostly used as side bearings, not as staircase kerns.


U+1D6FD

U+003A4

U+1D4CC

U+1D6B8

U+1D70C

In these figures you see a few glyphs from Cambria with staircase kerns and although we show them at a small size, you will notice that some kern boundaries touch the shape. As subscripts never go that high it goes unnoticed but it also shows that sticking to the lowest boundary makes sense.

We conclude that we can simplify these kerns, and just transform them into our (up to four) corner kerns. It is unlikely that Cambria gets updates and that other fonts become more advanced. One can even wonder if multiple steps really give better results. The risk of overlap increases with more granularity because not every pair of glyphs is checked. Also, the repertoire of math characters will likely not grow substantially, or include shapes that differ much from what we can look at now. Reducing these kerns to simple ones, that can easily be patched at will in a goodie file, has advantages. We could even simplify the engine that way.

6 Conclusion

So, how can we summarize the above? The first conclusion is that we can only get good results when we runtime patch fonts to suit the engine and our (ConTEXt) need. The second conclusion is that we should seriously consider to drop (read: ignore) most math font parameters, and/or to reorganize them. There is no need to be conforming, because these parameters are often not that well implemented (thumb in mouth). The third conclusion, or perhaps observation, is that we should get rid of the excessive use of italic correction, and go for our new corner kerns instead. Last, we can conclude that it makes sense to explore how we can use MetaPost to analyze the shapes in such a way that we can improve intercharacter kerning, corner kerns and maybe even, in a limited way, staircase kerns.

And, to come back to accents: very few characters need a top kern. Most can be handled with centered anchors, and we need tweaks for margins and overshoot anyway. The same is true for many other tweaks: they are there to stay.

This is how we plan to go forward:

- We pass no italic corrections in the math fonts to the engine, but instead we have four dedicated simple corner kerns, top and bottom anchors, and we also compensate for a negative left side bearing. We should have gone that route earlier (as a follow-up on a MkIV feature) but were still in some backward compatibility mindset.
- The LuaMetaTEX math engine might then be simplified by removing all code related to italic correction. Of course it hurts that we spent so much time on that over the years. We can anyway disable engine options related to italic correction in the ConTEXt setup. Of course the engine is less old school generic then but that is the price of progress.
- A default goodie file is applied that takes care of this when no goodie file is provided. We could do something in the engine, but there is no real need for that. We can simplify the mid-2022 goodie files because we have to fix fewer glyphs.
- If we end up needing italic corrections again (that is: backtrack) then we can use the (new) `\mccode` option code that can identity sloped shapes. But, given that ignoring the correction between sloped shapes looks pretty bad, we can as well forget about this. After all, italic correction was never so much about correcting italics, but more about anchoring scripts.
- Staircase kerns can be reduced to simple corner kerns and the engine can be simplified a bit more. In the end, all we need is true widths and simple corner kerns.
- We reorganize the math parameters and get rid of those that are not truly dependent on the font design. This also removes a bit of overlap. This will be done as we document.
- Eventually we can remove tweaks that are no longer needed in the new setup, which is a good thing as it also saves us some documenting and maintenance.

All this will happen in the perspective of ConTEXt and LuaMetaTEX but we expect that after a few years of usage we can with confidence come to some conclusions that can trickle back into the other engines so that other macro packages can benefit from a somewhat radically different, but reliable, approach to math rendering, one that works well with both old and new fonts.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University
Box 118
221 00 Lund
Sweden
`mickep (at) gmail dot com`

Patching Lucida Bright Math

Hans Hagen, Mikael P. Sundqvist

1 Introduction

During the last year we have been working on the typesetting of mathematics in ConTEXt LMTX. This system is using OpenType fonts, and in particular Unicode math fonts. In the last decade several such math fonts have been created, many of them by converting old fonts from the Type 1 format. Lucida Bright Math[1] is one of these fonts, though the designers, Charles Bigelow and Kris Holmes, made significant additions and changes when developing the OpenType version. It comes in two weights, regular and bold.

While working through the math engine we have been running tests with essentially all the freely available OpenType math fonts available, and we have noticed that, besides being different in the approach to italic corrections, kerning and metrics, they all come with small issues. This is not so surprising, given that the fonts typically have thousands of glyphs, many parameters, and the specification of the format is often vague.

We fixed many common font issues in so-called goodie files, and the patching takes place at runtime. (See the accompanying article in this issue for much more about this.) We have finally come to a point where we believe that we have a model where most of the fonts look OK, independent of whether they are old converted TEX fonts controlled by italic corrections or new fonts driven by staircase kerns. We consider Lucida Bright Math to be one of the better fonts, both in the sense that the design is beautiful and that we did not have to tweak it so much to get it look right.

Nevertheless, we found some flaws in the font, and reported a few of them on the Lucida mailing list. They were put on the list of corrections to be fixed for the next Lucida release. In the meantime, we began to discuss the possibility to do font fixes directly in the font editor FontForge. Combined with the possibility to debug with the available visual helpers in ConTEXt LMTX, we realized that we had a rather effective work flow for editing and fixing. The problem-solving part and the direct payoff when we could see things getting corrected live on screen also made the process a joy. Below we give an overview of the fixes we did. We hope that the Lucida users out there will benefit from these changes.

[1] Read more about this font at `tug.org/TUGboat/tb37-2/tb116bigelow-lucidamath.pdf`

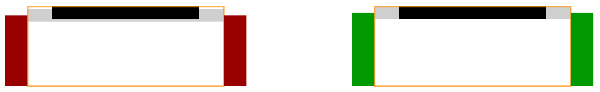
After the fixing we need fewer tweaks in ConTEXt for Lucida, but we also have to make sure that tweaks could be applied per font version, because even with TUG's unique update policy it might take a while before all users have the new version.

2 Correcting the math axis

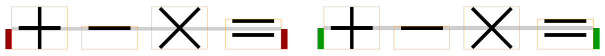
This is what Microsoft writes[2] about the math axis (our emphasis):

> "In math typesetting, the term axis refers to a horizontal reference line used for positioning elements in a formula. The math axis is similar to but distinct from the baseline for regular text layout. For example, *in a simple equation, a minus symbol or fraction rule would be on the axis*, but a string for a variable name would be set on a baseline that is offset from the axis. The axisHeight value determines the amount of that offset."

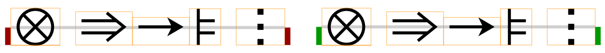
Let us look at the minus sign.[3]



The minus sign to the left is not centered vertically on the math axis. The value of the math axis in Lucida Math Bright has been set to 313, but the minus was centered on 325. At first we thought that this might have been a problem with the minus sign, but when looking at more glyphs, we realized that a large majority of the ones that one could argue should be aligned vertically around the math axis were in fact aligned to the height 325. We show below first some of the most common symbols



and some others, less often used



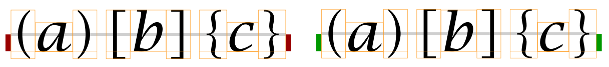
In fact, almost all symbols that one could argue should be placed vertically centered on the math axis are centered on 325. We conclude that the value 313 is not correct for this font, it should simply be 325. Thus, *we changed the math axis to 325.* This had some, not too big, consequences. We needed

[2] `learn.microsoft.com/en-us/typography/opentype/spec/math#mathconstants-table`

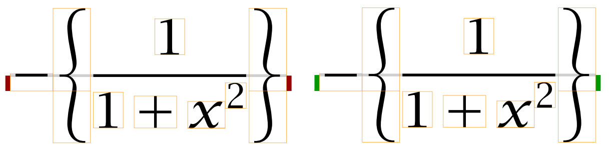
[3] Here, and in the continuation, we show in our examples the output before our edits (on the left or above) between a pair of red rules (dark gray on paper) and the edited version (right or below) between green ones (medium gray). The red and green boxes indicate the height of the math axis. The gray rule in the background is centered at the height of the math axis. Orange (light gray) boxes show bounding boxes of glyphs (with an extra line for the baseline, if the glyph has a non-zero height and depth). The glyph shape itself is black.

doi.org/10.47397/tb/43-3/tb135hagen-lucida

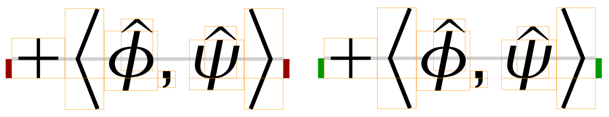
to adapt vertically the parentheses, since originally they were aligning vertically on the old math axis (i.e., they were correct before). We also needed to align integrals (some of them were in fact not centered at the math axis before).



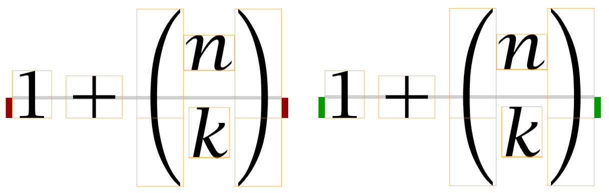
In practice this meant that many glyphs needed to be raised by 12 units in FontForge. We emphasize again that this was done to have a perfect vertical alignment with the minus and plus signs, and the other symbols that live symmetrically around the math axis. This is striking for the braces



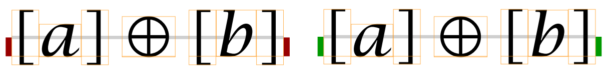
and for the angle brackets



but the difference is less obvious for the round parentheses

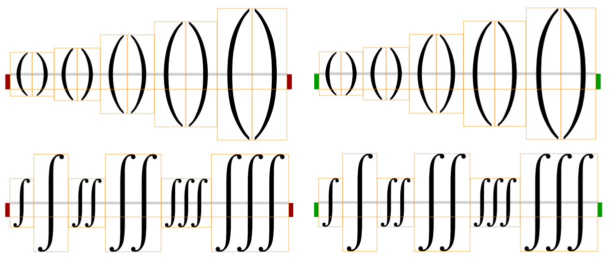


and the square brackets



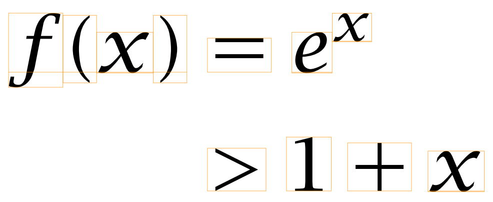
3 Aligning around the math axis

As mentioned, the different parentheses and similar glyphs were centered around the old math axis 313. We have shifted them up 12 steps so that they are now aligned with the corrected math axis. Below we give a few examples of glyphs that were shifted (all of them were of course reported to TUG).

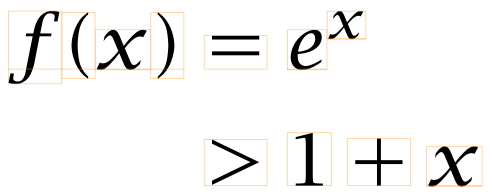


4 Modifying glyph sizes

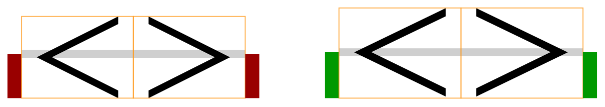
Our first mail to the Lucida mailing list was about the < and >. We noticed that their size was different from other similar relation symbols, like the equal sign =. This becomes a problem when one is aligning equations on different lines, since then one usually aligns on these characters, and if they are of different width, the result will not look good. This is how it could look in the old font:



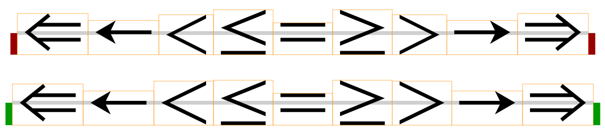
If you look carefully at the e and the 1, they are not horizontally aligned with each other, and the reason for that is simply that the bounding box of > is smaller than the bounding box of =. We added a temporary fix to ConTEXt (in the goodie file for Lucida) where we scaled these glyphs. But it is better to fix them in the fonts. After fixing the font, the example looks like this.



We not only changed the bounding box of the glyphs, but also scaled the glyph horizontally so that it had the same width as the equal sign (the glyph, not the bounding box!). We also scaled it vertically so that it became centered on the math axis. When that was done, we made sure that it ended up with the same side bearing as the equal sign.

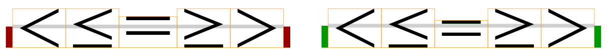


It might help to see these symbols together with some other similar ones.

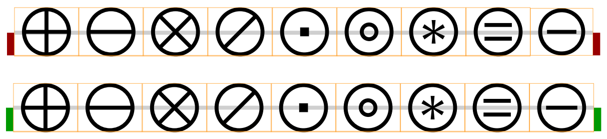


Lucida Bright Math also has some alternative, in fact smaller, versions of some glyphs. (They are the operators from the original Type 1 fonts; the font designers increased the operator size in OpenType based on their observations of usage and user requests, but some users prefer the original size, so both are available.) They are available by activating the ss03 style alternative, "small operators". Here, the problem is that the equal sign has no alternate, so the big version is used.

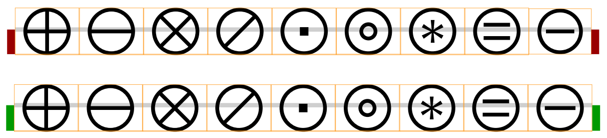
The previous less than and greater than fit well with the alternate stylistic set ss03. We thus added new slots at the end of the font for that purpose. Also, the equal sign lacked a version in ss03, so we added it. Note that the symbols in ss03 are not in general centered vertically around the math axis.



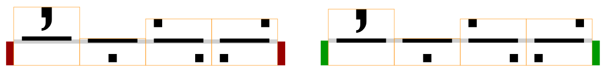
FontForge has a nice view where it is easy to browse the glyphs one by one with the position preserved. That makes it very clear when there are differences in following glyphs that should be similar. While browsing in this manner in the slots 0x2295 to 0x229D (various circles with decorations inside) we noticed that the last one had different dimensions than the other. This was the case both for the ordinary glyphs



and for the smaller variants in ss03.

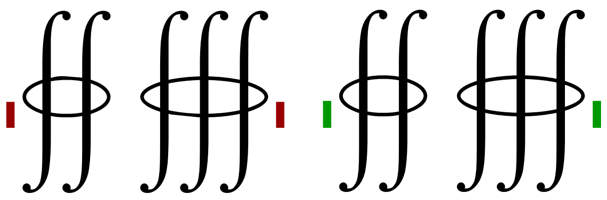


As you can see, the last one was a bit small, and is fixed. One more such symbol that stood out, 0x2A29, the minus sign with a comma on top (who uses that?):



5 Artifacts in the integral glyphs

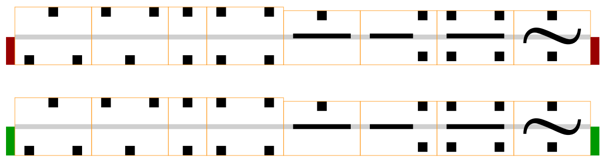
While going over the shifting of the parentheses and integrals, we noticed two glyphs that had defects. It was the display versions of 0x222F (`\oiint`) and 0x2230 (`\oiiint`), and in both cases it was the oval part of the symbol that was incorrect.



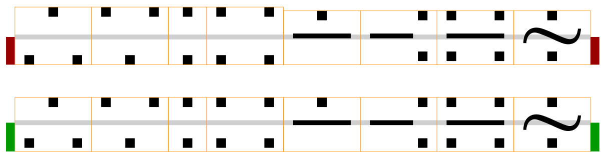
We realized that the glyphs had some extra points added that messed them up. We simply removed the extra points, made some point a corner point and tuned the control points to agree with the other corresponding ones in the glyph.

6 Dots, dots, dots

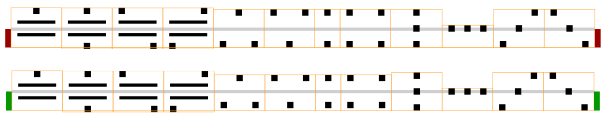
We also found some inconsistent combinations of glyphs that include dots (or, squares, as they are in Lucida). If we look at 0x02234 to 0x0223B, for example, we see that first four are different from the final four.



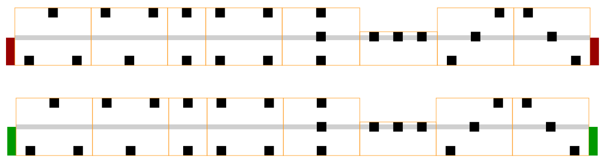
We first decided to make the first four of these adopt the spacing of the last four. It was mainly the ratio symbol (0x2236, the third in the list) that made us decide that, since we agreed that the two squares in it are simply too far away from each other (remember, this symbol is used in the `\colon` construction). We then ended up with this.



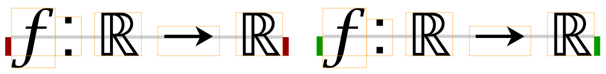
As there are so many symbols, we eventually found out that our local changes introduced new inconsistencies.



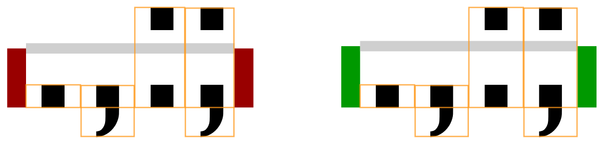
We therefore decided to ditch our first changes, but instead of throwing them away, we added the glyphs as a new set of style alternates, ss06. Thus, these glyphs stay unchanged (note that the new choice of math axis also agree with them):



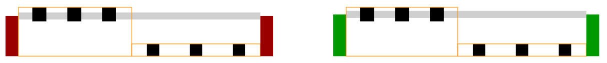
We are still not completely happy with the ratio, and prefer the version from the newly added alternate set (below shown to the right).



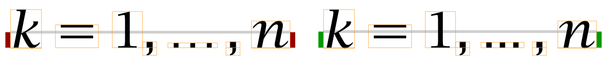
While discussing the ratio, we looked also at the normal colon (0x3A) and semicolon (0x3B) characters, and we noticed that their side bearings were not symmetrical. We thus fixed that, and made them consistent with the period and the comma.



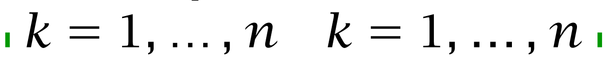
Finally, we also saw that the ellipses on the baseline (0x22EF) are wider spaced than the ones on the math axis (0x2026). We decreased the right side bearing of 0x22EF so that became consistent with similar constructions.



We added yet another style alternative, ss07, with a version of 0x22EF with the small squares spaced in the same way as the other similar glyphs.

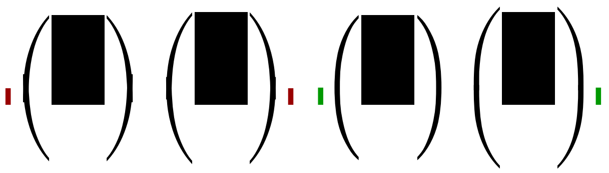


One can still question the fact that the squares in the ellipses are smaller than the squares in the comma and the period.

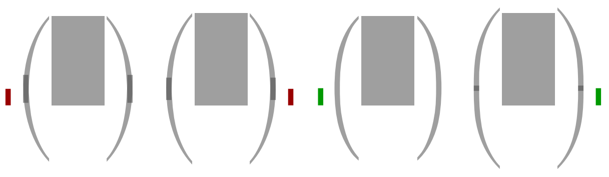


7 Extensible recipes

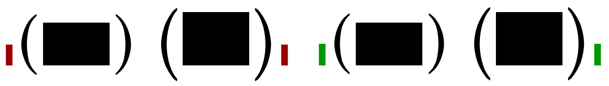
We did not want to touch the extensible recipes. But then we saw that the top and bottom pieces of the round parentheses, when just too large for the largest variants, clash into each other with bad results.



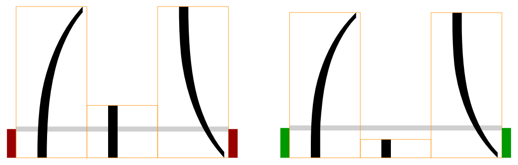
The problem becomes more apparent if we use transparent colors.



Note that in the fixed version, the largest variant is still used in the left example, while the first extensible is used in the right one. This means, and that is unavoidable, that the parentheses are slightly taller than the content. This can of course also happen when we use the variants.



So, how should the extensible recipe be built? Let us look at the left parenthesis. It consists of three parts, the “left parenthesis upper hook” (0x239B), the “left parenthesis extension” (0x239C) and “left parenthesis lower hook” (0x239D).



There is a table in the font, a recipe, that decides how the extensible is to be built. For the left parenthesis, the first and the third are always used once, and then there can be as many middle ones as needed (including zero). Looking at Table 1 we see that the size of the glyphs are 1648 (top and bottom) and 570 (the middle one). Since the top and bottom glyphs are needed, the minimum height plus depth will be $2 \times 1648 = 3296$. Or, that is what one could imagine. But there is a font parameter `MinConnectorOverlap`, set to 40 in Lucida Bright Math. It is supposed to be the minimal overlap of the glyphs. This means that the true minimal height is 3256. This can be compared with the height plus depth of the largest variant of the left parenthesis, $1686 + 1061 = 2747$.

Table 1: Original extensible values.

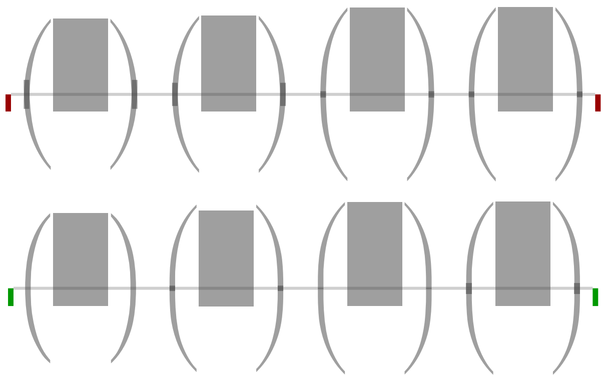
`Glyph`	`Extender`	`StartLen`	`EndLen`	`FullLen`
uni239D	false	549	549	1648
uni239C	true	190	190	570
uni239B	false	549	549	1648

The biggest problem is not the values in the table, but the fact that the top and bottom parts do not have ends that can overlap aesthetically. No matter how little we overlap, the pieces will not fit perfectly with each other, or with the rectangular middle piece, since they are not rectangular themselves at the end. Thus, we wanted to add a rectangular part to the first and the last pieces. But then we got another problem. If we just added a rectangular piece, the glyphs would be too large, and the extensibles would kick in too late. After some trial and error, we decided to scale the original top and bottom parts just slightly, and then to add a rectangular piece of height 100. We changed the height of the extensible part to 200. Inspired by the TeX Gyre fonts, we ended up with the values in Table 2.

Table 2: Updated extensible values.

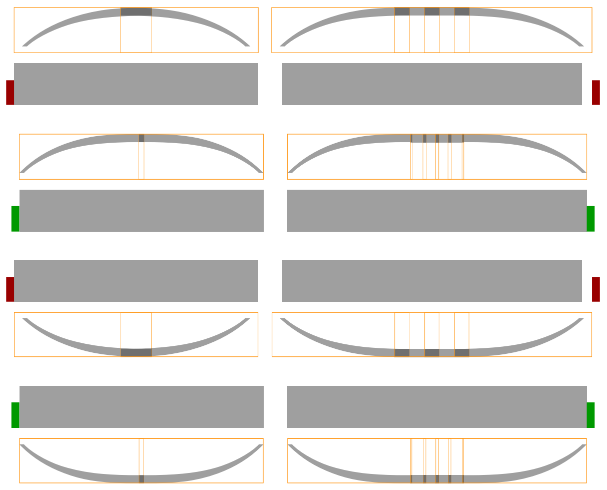
`Glyph`	`Extender`	`StartLen`	`EndLen`	`FullLen`
uni239D	false	0	100	1583
uni239C	true	200	200	200
uni239B	false	100	0	1583

The top and bottom pieces are allowed to overlap by 100 units with each other (or with the middle piece), exactly the size of the added rectangle. For maximum flexibility, the middle piece is allowed to overlap 200 in both directions. Let us look at a few examples.

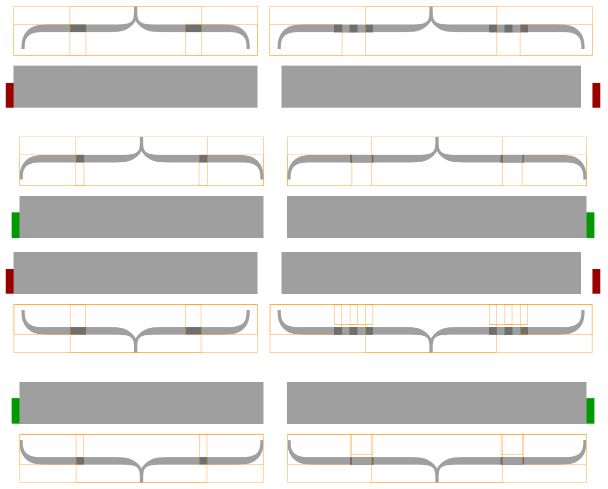


In the first case (to the left), the content is high enough to trigger the extensible in the unfixed font, but not in the fixed one. In the next, the content is precisely sufficiently high to trigger the extensible also in the fixed font. Observe that the parentheses are slightly bigger than the content. The grayer area shows the overlap, and it corresponds approximately to the value 100 in Table 2. If we increase the content a bit more we still get no middle piece, but the overlap is now very small. The size now fits the content well. Finally, with slightly larger rule, the new version adds a rectangular middle piece. If you look carefully, you will see that almost all of it is overlapping with the other two pieces. The unfixed font still has no middle piece.

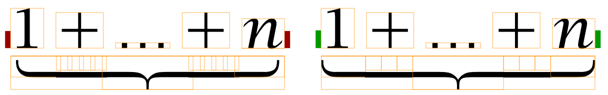
A similar situation is present for symbols that scale horizontally, and in particular for the parentheses. We decided to scale equally as much as for the vertical parentheses, and then also add a rectangular piece.



An analogous change was made to the horizontal up and down braces.

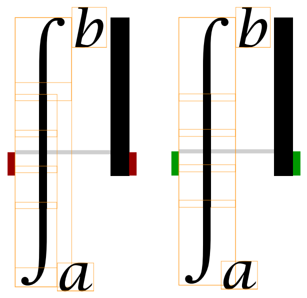


The attentive reader may notice that the side bearings of the glyphs are set to zero. We could not find any other math font with a non-zero side bearing for these glyphs. That makes sense, since they only complicates the calculations.



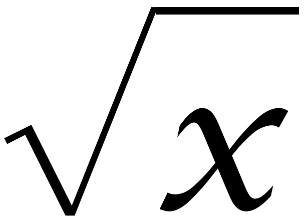
A character in an OpenType font can have two variant lists, two extensible recipes and two extra italic corrections, meant to support both horizontal and vertical extensibles. Only a few fonts have these extra italic corrections set, and we have only observed them on integral signs, and we haven't seen corrections set on horizontal extensibles at all. The (vertical) italic correction is used for positioning the subscript and limits (on n-ary operators).

Lucida is one of those fonts that has an extensible integral. Although the LuaMetaTEX engine will use the maximum width of a snippet, to be coherent with the other fonts with extensible integrals, we have made sure that in the updated Lucida all snippets have the same width, as shown in the example below (width of middle and bottom pieces now matches width of top piece). We did not reset the corrections on individual snippets, because these are ignored anyway.

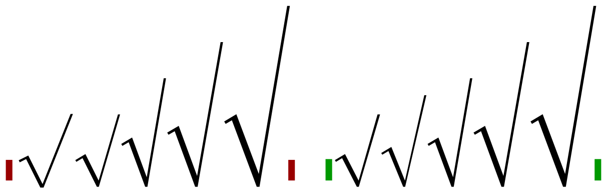


8 The radical symbol

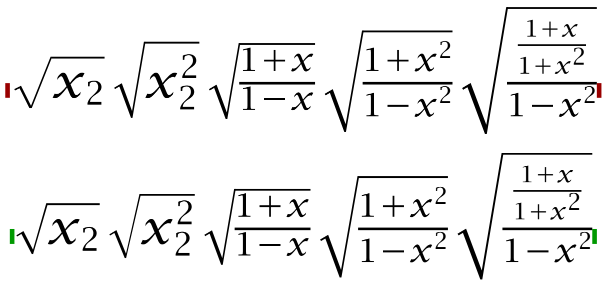
Our last stop is the radical symbol, and the reason to stop here can be seen in the following simple expression:



The radical symbol and the horizontal rule do not fit together. When we were looking at the variants of the radical symbol, we became surprised by two things.



We first noted that the base glyph and the variants look different. Next, we realized that the first variant will probably never kick in, since it has the same size as the base glyph. We decided to move the first variant into the base glyph, to get a consistent look, and to modify the size of the first variant slightly. In the following set of examples we see that the new size is used (second from left) in the fixed version, while we get the slightly larger radical for the unfixed one.



Regarding the mismatch between the radical and the horizontal rule, this turned out to not be a problem in the font (except for the old radical base character). It was instead related to a backend related snapping feature in ConTEXt, used to prevent loading fonts in too many sizes due to rounding errors.

This is comparable to cases where TEX's scaling of 1000 means 1.000 with three digit precision, although that often goes unnoticed because it happens consistently in the whole document. The mismatch doesn't happen when we operate in PostScript points (bp) that are natural to both OpenType fonts and PDF, but it does when we use TEX points (pt) which means that when going to PDF's points we lose some accuracy. The difference between 0.9963 and 0.9954 is noticeable to the sensitive eye when you blow up these composed glyphs for testing, and we can't tolerate that, can we? So we now go for more precision at the cost of (possibly) some font sizes. There is currently an experimenting mode in ConTEXt, the compact font mode, where any font is loaded just once, anyway, but that has to be discussed elsewhere.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University
Box 118
221 00 Lund
Sweden
mickep (at) gmail dot com

Ventrella's terdragon in METAPOST

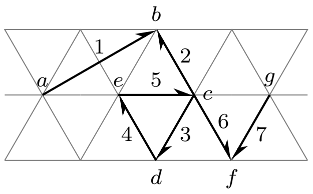
Linus Romer

Abstract

This article shows how to create a vector graphic similar to the terdragon described by Ventrella (2019) in METAPOST.

1 Generating the fractal path

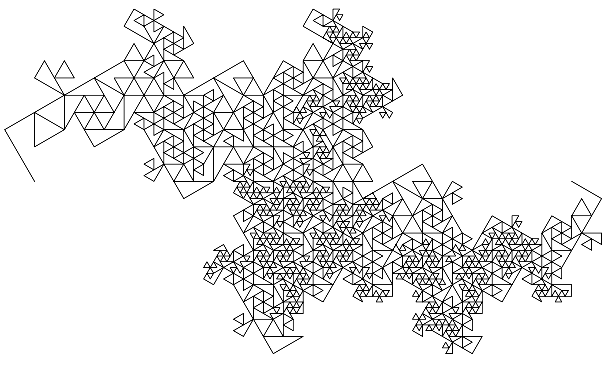
The fractal used for the terdragon is based on the following recursive replacement pattern on a triangular grid. Reverse arrows indicate segments that are rotated by 180°:



This pattern can be implemented as a recursive macro with recursion depth n:

```
vardef dragon(expr a,g,n) =
 save p,b,c,d,e,f; path p;
 if n > 0:
  pair b,c,d,e,f;
  e = 1/3[a,g]; c = 2/3[a,g];
  b = .5[a,g]+sqrt(3)/2*((c-e) rotated 90);
  d = c+e-b; f = g+e-b;
  p = dragon(a,b,n-1)
      & reverse dragon(c,b,n-1)
      & dragon(c,d,n-1) & dragon(d,e,n-1)
      & dragon(e,c,n-1) & dragon(c,f,n-1)
      & reverse dragon(g,f,n-1);
 else:
  p = a -- g;
 fi
 p % the returned path
enddef;
```

Using `draw dragon((0,0),(300,0),4);` you will get the following figure:



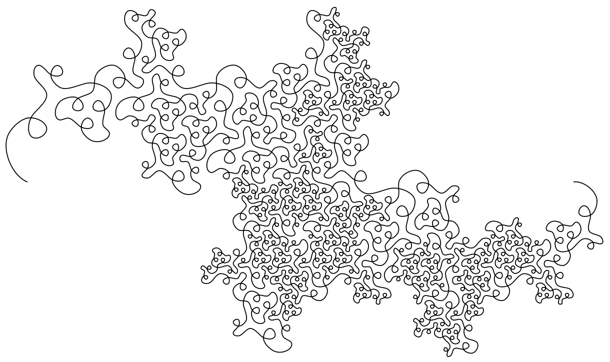
2 Rounding the vertices to curves

In the previous picture many of the vertices are revisited at different travel times on the path. This self-contacting behaviour can be avoided in different ways. In order to keep the number of path points small, weighted averages between neighbour vertices are used to draw smooth Bézier curves through them:

```
def roundcorners expr p =
  point 0 of p
  for i = 1 upto length(p)-1:
   .. tension 1.2
   .. ( .64*(point i of p)
       + .18*(point i-1 of p)
       + .18*(point i+1 of p) )
  endfor
  .. tension 1.2 .. point length(p) of p
enddef;
```

The tension 1.2 and the weights 0.64 and 0.18 are somewhat arbitrary. The following figure is produced by:

```
draw roundcorners dragon((0,0),(300,0),4);
```



It can now be observed that this terdragon variation is self-crossing.

3 Stroking the path dynamically

A dynamic stroking of the path is achieved by making the stroke width proportional to the distance between the points:

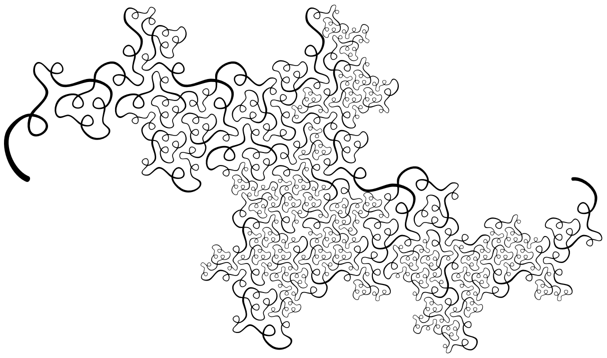
```
vardef dynamicdraw expr p =
 save n,l,r,s;
 pair l[],r[];
 numeric n; n = length(p);
 for i = 0 upto n:
  l[i] - r[i]
  = unitvector(direction i of p rotated 90)
    * ( length(point max(1,i) of p
              - point max(i-1,0) of p)
      + length(point min(n,i+1) of p
      - point min(i,n-1) of p) )
    * .08;
  r[i] + l[i] = 2*point i of p;
```

doi.org/10.47397/tb/43-3/tb135romer-terdragon

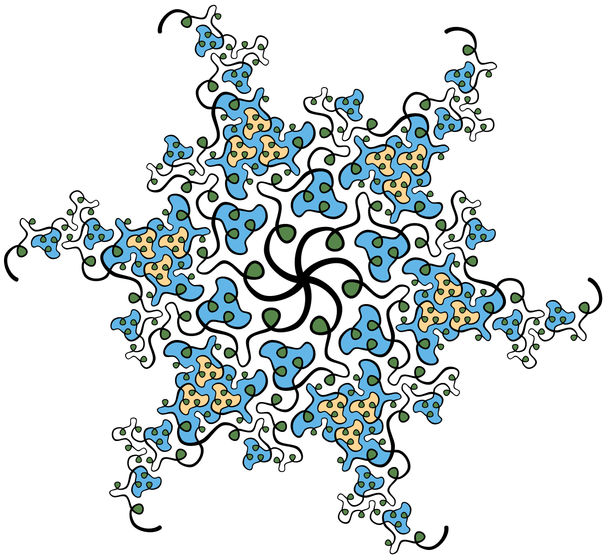
```
 endfor
 fill l[0]{direction 0 of p}
 for i=1 upto n:
  .. tension 1.2 .. l[i]{direction i of p}
 endfor
 for i=n downto 0:
  .. tension 1.2 .. r[i]{-direction i of p}
 endfor .. tension 1.2 .. cycle;
enddef;
```

The width scale 0.08 may be changed according to one's personal taste.

Replacing `draw` by `dynamicdraw` changes the last figure to the following:



If desired, a vector drawing program like *Inkscape* may be used to fill the enclosed areas with colour. After rotating a coloured (though grayscaled for the printed *TUGboat*) terdragon with recursion depth 3 six times, with respect to its left end, you will get the following figure:



For more information about the terdragon, please see the references, among numerous other books and articles.

References

Ventrella, Jeffrey. *Brainfilling Curves — A Fractal Bestiary*. Eyebrain Books, 2012.

Ventrella, Jeffrey. "Portraits from the Family Tree of Plane-filling Curves". In *Proceedings of Bridges 2019: Mathematics, Art, Music, Architecture, Education, Culture*, edited by S. Goldstine, D. McKenna, and K. Fenyvesi, pages 123–130, Phoenix, Arizona. Tessellations Publishing, 2019. Available online at `archive.bridgesmathart.org/2019/bridges2019-123.pdf`.

◇ Linus Romer
Ahornstrasse 8
Uznach, 8730
Switzerland

An introduction to GNU 3DLDF

Laurence Finston

Abstract

This article is an introduction to GNU 3DLDF. GNU 3DLDF is a package for three-dimensional drawing with METAPOST and METAFONT output. It implements a language based on the METAFONT language with many additional data types and operations. It is designed for general technical drawings and a particular focus is intersections of geometrical figures.

Introduction

GNU 3DLDF is a package for three-dimensional drawing with METAPOST and METAFONT output. It implements a language based on the METAFONT language with many additional data types and operations.

METAFONT is a program for font design; its output is run-length encoded (i.e., compressed) bitmaps which may be converted to a form suitable for display on computer monitors or for printing. Since METAFONT was completed in 1984, scalable fonts have become the de facto standard, so METAFONT's bitmap format, though still usable, and despite the nice features of METAFONT fonts, unfortunately may be considered largely obsolete. In current TEX distributions, PostScript or OpenType versions of Knuth's Computer Modern fonts, originally programmed in METAFONT, are used by default.

METAPOST is a modified version of METAFONT that produces output in the form of PostScript code. While METAFONT is specifically designed for the purpose of font design, METAPOST may be used for technical drawings in general. However, while it includes some features not present in METAFONT, it has not diverged very far.

A first example

The following example is intended to give a first impression of 3DLDF. It shows a circle rotated and shifted in 3D space while at the origin, a set of arrows point in the directions of the positive and negative x-, y- and z-axes. The drawing is projected using the *perspective projection*, so that the arrows representing the z-axis are foreshortened.

Save the following 3DLDF code in a file named `minimal.ldf` (downloadable reference given at the end):

```
verbatim_metapost "prologues := 3;"
   & "outputtemplate := \"%j%3c.eps\";";
numeric frame_wd, frame_ht;
path frame;
frame_wd := frame_ht := 2cm;
frame := (-frame_wd, -frame_ht)
 -- (frame_wd, -frame_ht)
 -- (frame_wd, frame_ht)
 -- (-frame_wd, frame_ht)
 -- cycle;
pen medium_pen;
medium_pen := pencircle
   scaled (.375mm, .375mm, .375mm);
pickup medium_pen;
focus f;
set f with_position (-20cm, 20, -50)
  with_direction (-20cm, 20, 10)
  with_distance 70;

beginfig(1);
  circle c;
  c := (unit_circle scaled (1cm, 0, 1cm)
     rotated (50, 30, 0))
        shifted (2.25cm, .75cm, 2cm);
  draw frame shifted (1cm, 1cm);
  draw c;
  label("$c$", get_center c);
  drawdblarrow (-.5cm, 0, 0) -- (.5cm, 0, 0);
  drawdblarrow (0, -.5cm, 0) -- (0, .5cm, 0);
  drawdblarrow (0, 0, -.5cm) -- (0, 0, .5cm);
  label.top("$x$", (.5cm, 0));
  label.rt("$y$", (0, .5cm));
  label.lft("$z$", (0, 0, .5cm));
endfig with_focus f;
verbatim_metapost "end";
end;
```

Run the following commands:

```
3dldf minimal.ldf
mpost -numbersystem="double" minimal.mp
```

This is the result (`minimal001.eps`):

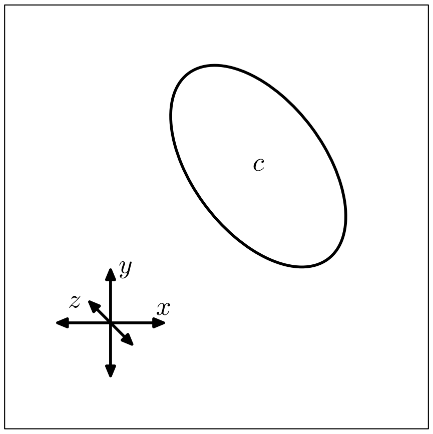


Fig. 1.

doi.org/10.47397/tb/43-3/tb135finston-3dldf

It is a "standalone" Encapsulated PostScript file (EPS) that can be viewed in a PostScript viewer such as Ghostview or Evince (Document Viewer) or included in a TeX file by means of the `\epsffile` macro defined by the `epsf.tex` file.

Save the following TeX code in `minimal.tex`:

```
\input epsf
\nopagenumbers \headline={}
%% DIN A4 Portrait
\special{papersize=210mm, 297mm}
\hsize=210mm \vsize=297mm
\parindent=0pt \parskip=0pt
\baselineskip=0pt
\advance\voffset by -1in
\advance\hoffset by -1in
\advance\hoffset by .75cm
\advance\voffset by 1cm
\def\epsfsize#1#2{#1}
\leftline{\epsffile{minimal001.eps}}
\bye
```

Run the following commands to create a PDF file containing the figure:

```
tex minimal.tex
dvipdfmx minimal.dvi
```

MetaPost output can also be read directly by pdfTeX and LuaTeX (the `{1}{1}` are scale factors):

```
\input supp-pdf
\convertMPtoPDF{minimal001.eps}{1}{1}
```

Motivation

In the 1980s I learned to make perspective drawings in the traditional way by hand, which is a tedious and error-prone procedure. In 1991, I had access to the computer-aided design (CAD) software AutoCAD and was excited by the possibility of using it to make "three-dimensional" drawings. Simultaneously, I had begun to learn to program in the computer language C.

Like a number of other interactive computer programs, such as Emacs or GIMP, AutoCAD implements a *command interpreter* in a language based on LISP. In the case of AutoCAD, it's called "AutoLISP". I soon discovered that I preferred "programming" my drawings rather than constructing them by pointing and clicking with a mouse.

In 1996, after several years of experience with TeX, I first used METAFONT for a project involving TeX for which I required some special characters. I enjoyed it and quickly discovered METAPOST and began using it for drawings. Soon I had the idea that it would be nice to have a 3D version of METAPOST, especially since AutoCAD was (and still is) very expensive and at the time required special equipment.

Some time later, I learned C++ for a job. At first I was reluctant but then I discovered that I liked the language and since then, had it in the back of my mind that I'd like to use it for some future project.

In 2002, I finally had the opportunity to do this and began to realize my idea of a "3D METAPOST" using C++. In 2003, 3DLDF was accepted into the GNU Project of the Free Software Foundation and I have continued to develop it since then.

The relationship of 3DLDF to METAFONT and METAPOST

As stated above, 3DLDF implements a language based on the METAFONT language. A 3DLDF program should therefore have a familiar "look and feel" to users of METAFONT and/or METAPOST. However, the implementation of 3DLDF has nothing to do with that of METAFONT or METAPOST. The latter were originally programmed with `WEB` in Pascal under the constraints on computer hard- and software that applied in the late 1970s to the mid 1980s. The "official" distributions of METAFONT and METAPOST use a version converted from Pascal to C using `web2c` (`https://tug.org/web2c`). However, the new versions remain close to the originals and have *not* been rewritten to reflect changes in computer hard- and software since the time when they were created. METAPOST only has added a feature enabling arithmetical calculations with a higher precision than in the original [4, Appendix A, "High-precision arithmetic with MetaPost", p. 78].

METAFONT implements a *command interpreter* or just *interpreter*. There are many programs of this type and two tasks they require are *lexical scanning* or just *scanning* and *parsing*. METAFONT implements these functions with hand-written, optimized code intended by their author, Donald Knuth, to demonstrate the "Art of Computer Programming" (the title of his *magnum opus*, [6]) by efficiently solving a complex task under very strict constraints with respect to storage space and execution time.

While I can appreciate METAFONT as a work of technical mastery or even art, I don't consider it an example of the best way to write a computer program in the 2000s or 2020s, when the constraints that applied when METAFONT was developed have simply ceased to exist and when computers and programming tools can be found on nearly every desktop or even in most people's pockets.

For this reason and as a matter of practicality, I've programmed 3DLDF using C++ along with the very comprehensive C++ Standard Library and other software libraries such as the GNU Scientific Library and the pthreads library. For parsing, I use the package GNU Bison. One of the files Bison produces is a text document describing the grammar rules of the language implemented by the parsing function in Backus-Naur format. This is familiar to readers of *The TEXbook* and *The METAFONTbook*, where Knuth uses it to describe the grammar rules of the TEX and METAFONT languages, respectively.

For the main scanner, I do not use the Flex package often used for this purpose, although I do use it for other tasks within 3DLDF. The reason is that 3DLDF attempts to duplicate METAFONT's scanning procedure which operates according to a different principle than Flex. It is also particularly simple to implement as it requires only a single token of "lookahead" [7, ch. 6, "How METAFONT Reads What You Type", p. 49ff.].

If I were to have an idea how to perform a particular programming task in a new and more efficient manner, I believe it would be more useful if I were to contribute it to an existing software library or make it available in some other way, rather than simply incorporating it into a "monolithic" program. In fact, 3DLDF is implemented as a shared or unshared library and linked with a file ("compilation unit") containing a `main` function. The library may just as easily be linked with other `main` functions.

Main differences between METAFONT, METAPOST and 3DLDF

Since METAPOST is a development of METAFONT and remains so close to it, in the following I will generally refer only to METAFONT and in most cases, what I say will apply equally to METAPOST, unless otherwise noted or it refers to features not present in METAPOST, such as those involving digitization [7, ch. 24, "Discreteness and Discretion", p. 195ff.].

If, on the other hand, I refer to METAPOST, it will be because I'm referring to features specific to METAPOST and not present in METAFONT.

Most of the differences between METAFONT and 3DLDF are consequences of the addition of the third dimension. Other differences are due to the fact that METAFONT was specifically designed for the needs of type designers while 3DLDF is intended for general technical drawing. Finally, some are due to my own personal preferences.

A difference one must bear in mind is that METAFONT, METAPOST and 3DLDF each have different *canonical units*: In METAFONT, they are pixels, in METAPOST PostScript points, a.k.a. "big points" (**bp**) and in 3DLDF they are centimeters (**cm**). It is worth noting that in TEX, there are no canonical units and it will cause an error if a dimension is specified without units:

```
\dimen0=5\relax
! Illegal unit of measure (pt inserted).
[...]
? h
Dimensions can be in units of em, ex, in, pt,
pc, cm, mm, dd, cc, bp, or sp; but yours is
a new one!
```

`\relax` is needed because otherwise TEX will wait for further input containing the dimension specifier.

Equations and assignments. METAFONT supports programming in a "declarative" rather than an "imperative" style [7, p. 87]. This idea would appear to have been "in the air" at the time Knuth created METAFONT. However, in my opinion, like the New Math, it has not stood the test of time [2, pp. 1–2].

While for METAFONT, Knuth expresses a preference for the use of *equations* rather than *assignments*, unfortunately this is currently not possible for 3DLDF. METAFONT is able to keep track of dependencies and solve equations once sufficient data is available [9, ch. 28, "Dynamic linear equations", §585; ch. 29, "Dynamic nonlinear equations", §618] and [10].

I would like to implement this feature, but at the present time I don't know how to go about it nor whether it would be possible with 3D data. Nor do I consider this to be a priority. However, since I may yet do so, the `=` operator is reserved for equations, as in METAFONT, and `:=` must be used for assignments.

3DLDF does implement "declarative" forms of operations, such as transformations:

```
path q;
q := (0, 0, 0) -- (1, 1, 1)
   rotated (10, 15, 20);
```

However, for users who aren't afraid of hurting the computer's feelings, corresponding "imperative" operations are available as well:

```
rotate q (10, 15, 20);
```

In addition, 3DLDF implements the *operators* for assignment plus an arithmetical operation `+=`, `-=`, `*=`, and `/=` for different variable types, as appropriate. For example, all of them are available for **numeric**s, `+=` for vector-type variables (see "Vector-type objects", p. 328) and `*=` for applying transformations to **point**s, **path**s, etc.:

```
point p; p := (1, 2, 3);
point_vector pv; pv += p;
transform t;
t := identity scaled (3, 4, 5);
pv0 *= t;
show pv0;
```

results in:

```
point:
World coordinates:
(3.0000000, 8.0000000, 15.0000000, 1.0000000)
```

These “imperative” operators make it possible to avoid clumsy constructions such as `q := q rotated (10, 20, 30)` (which do also work, however).

Incidentally, the operators for assignment plus an arithmetical operation described above break the parsing rules of METAFONT. However, they were easy to implement and have never caused any problems.

Projections. For a 3D graphics program to be useful, the objects for which three-dimensional data are stored must be *projected* onto a two-dimensional plane for display on a computer screen or printing. 3DLDF implements two kinds of projection: parallel projection onto one of the major planes (x-y, x-z or y-z) and the *perspective projection.* In fact, it’s possible to project a drawing onto an arbitrary plane by transforming the desired plane so that it comes to lie in a major plane, transforming the objects to be drawn in the same way, and then projecting them onto the latter plane. Other projections of interest, but not yet implemented, are projections onto a cylinder, sphere or other curved surfaces and the Mercator projection and other projections used for maps.

Though parallel projections are extremely useful, even for 3D drawings, it is the *perspective projection* that is most closely associated with 3D graphics. It essentially simulates the effect of instantaneously photographing a scene with a camera or viewing it with one immobile eye. The result of the perspective projection is as if a line were drawn from each point in a scene to a *focus* represented by single point (the camera lens or the lens of an eye) and the intersection of this line with a plane (the *plane of projection* or *picture plane*). This plane may be imagined to be between the focus and the scene (the normal procedure), behind the scene or behind the focus, in which case the image appears upside-down, as in a camera obscura or the retina.

In addition to the position of the focus, the direction of view and the distance from the focus to the plane of projection must be specified and the “upwards” direction calculated. In 3DLDF, this data is stored in an object of type **focus**. To change the upwards direction, the focus may be rotated about the line from the position through a point lying in the direction of view from the position.

```
focus f; set f with_position (-10cm, 20, -50)
       with_direction (-10cm, 20, 10)
       with_distance 70;
show f;
⟶
focus:
position:
World coordinates:
(-10.0000000, 20.0000000, -50.0000000, 1.0000000)
direction:
World coordinates:
(-10.0000000, 20.0000000, 10.0000000, 1.0000000)
up:
World coordinates:
(-10.0000000, 21.0000000, -50.0000000, 1.0000000)
distance == 70.00000000.
   axis == z
angle == 0.00000000
```

Pairs and points

The most basic “drawable” object type in METAFONT is the **pair**, which is used to represent a point in the plane. It consists of two numerical values, an x- and a y-coordinate:

```
pair p; % METAFONT
p = (1cm, 2cm);
```

In 3DLDF, **pair** is replaced by the object type **point**, which is used to represent a point in three-dimensional space:

```
point p; % 3DLDF
p := (1cm, 2cm, 3cm);
```

From the point of view of a user, a **point** has three coordinates, **x**, **y** and **z**. However, in fact, **point**s have an additional fourth coordinate, namely **w**. Such a set of coordinates is called *homogeneous*. The w-coordinate is normally 1 but will usually be $\neq 1$ when the **point** is projected using the perspective projection, as described in the previous section. In addition, the fourth coordinate is needed in order to be able to multiply the **point** with a 4×4 matrix, which shall occupy our attention below.

In 3DLDF, **point**s have four sets of coordinates, “world”, “perspective”, “user” and “view”, of which currently only “world” and “perspective” are in use.

METAFONT’s **pair**s and 3DLDF’s **point**s are also used to represent the *difference* between two points in (2D or 3D) space, that is, the result of subtracting one point from another:

```
pair a, b; %% METAFONT
show a - b;
>> (-xpart b+xpart a,-ypart b+ypart a)
a = (2, 3);
b = (1, 2);
show a - b;
```

```
>> (1,1)
```

```
point a, b, c; %% 3DLDF
a := (2, 3);
b := (1, 2);
c := a - b;
show c;
⟶
point:
World coordinates:
(1.0000000, 1.0000000, 0.0000000, 1.0000000)
```

The result of subtracting one point from another is called a *vector*, which has a *magnitude* and a *direction*, but no location in space. **pair**s in METAFONT and **point**s in 3DLDF are used to represent both points in (2D or 3D) space and vectors, and the same object may be interpreted as a point or a vector, depending on circumstances.

In 2D, the magnitude is the distance to a point with the same x- and y-coordinates, i.e., $\sqrt{x^2+y^2}$ and similarly in 3D, with the added z-coordinate, i.e., $\sqrt{x^2+y^2+z^2}$ and the direction is that indicated by a line from the origin to that point. Such vectors are of great importance in 3D graphics. They should not be confused with the data type **vector** in C++ or other uses of the term. A *unit vector* may be created by dividing the x-, y- and z-coordinates of a **point** by the magnitude (input will be directly followed by output from now on):

```
point p[];
p0 := (2, 2, 2);
n := magnitude p0;
show n;
>> 3.4641
p1 := (2/n, 2/n, 2/n);
show p1;
point:
World coordinates:
(0.5773503, 0.5773503, 0.5773503, 1.0000000)
show magnitude p1;
>> 1
```

Since unit vectors are needed so often, 3DLDF implements the **unit_vector** operator for this purpose:

```
p2 := unit_vector p0;
show p2;
point:
World coordinates:
(0.5773503, 0.5773503, 0.5773503, 1.0000000)
show magnitude p2;
>> 1
```

In METAFONT programs, instead of using **pair**s, it is the convention to use **z** for representing points, e.g., `z = (4pt, 5pt);`. This convention is implemented by means of a **vardef** macro [7, p. 277]:

```
vardef z@#=(x@#,y@#) enddef
```

There is no correspondence to this macro in 3DLDF. For one thing, since **point**s already have a **z**-coordinate, **z** would be a poor choice for the default name for points and there is no other obvious choice, since "**p**" would be equally appropriate for **path**s and possibly even for **polygon**s or **parabolæ** (to say nothing of **polyhedra** or **paraboloid**s). Therefore, in 3DLDF, explicitly declared **point**s must always be used to represent points in space (and vectors).

Transformations

METAFONT implements the *transformations* shifting (translation), scaling and rotation by means of the object type **transform**, which consists of 6 numerical elements:

```
transform t; %% METAFONT code
show t;
>> (xpart t,ypart t,xxpart t,xypart t,
yxpart t,yypart t)
```

A **pair** may be "transformed" like this:

```
pair a, b;  %% METAFONT code
a = (1, 2);
transform t;
t = identity scaled (2, 4);
show t;
>> (0,0,2,0,0,4)
b = a transformed t;
show b;
>> (2,8)
```

The *identity* transformation is needed as a starting point for other transformations and looks like this:

```
show identity; %% METAFONT
>> (0,0,1,0,0,1)
```

In 3D, transformations are represented by 4×4 matrices of numerical values, which are called *transformation matrices*. Therefore, even if the perspective projection wasn't needed, **point**s would have to have 4 coordinates in order that they may be multiplied with transformation matrices. The identity matrix in 3DLDF looks like this:

```
show identity;
transform:
      1       0       0       0
      0       1       0       0
      0       0       1       0
      0       0       0       1
```

In 3DLDF, the syntax with **transformed** is supported. However, it also implements the operation "multiplication with assignment", so that the operator `*=` may be used to perform a matrix multiplication on a **point**:

```
point p;
p := (1, 2, 3);
transform t;
```

```
t := identity scaled (2, 3, 4);
show t;
transform:
      2       0       0       0
      0       3       0       0
      0       0       4       0
      0       0       0       1
p *= t;
show p;
point:
World coordinates:
(2.0000000, 6.0000000, 12.0000000, 1.0000000)
```

Paths

While **pair**s in METAFONT and **point**s in 3DLDF are the basic building blocks of drawings, **path**s are an essential element of any font or technical drawing: Without **path**s, all you have is a scatter plot.

In METAFONT, **pair**s and **path**s are the only "drawable" types. Font design tends to use free-form curves, so Knuth clearly didn't consider it necessary to define data types for algebraic curves, for example. plain METAFONT does define **quartercircle**, **halfcircle**, **fullcircle** and **unitsquare**, but these are constants of type **path**. METAFONT doesn't store any additional information about them, such as their centers or radii. It also defines **superellipse** as a macro.

3DLDF, on the other hand, is intended for general technical drawing. Many technical drawings consist solely of straight lines and where curves are used, by far most often they are circles, circular arcs, ellipses and elliptical arcs. Ellipses play a special role because circles appear elliptical in perspective. A common tool for technical drawing is (or was) stencils for drawing perspective ellipses.

Other curves occur frequently in drawings for special purposes: helices (non-planar) for representing screw threads, epi- and hypocycloids for gear teeth [1, pp. 53–55], parabolæ for trajectories, catenaries for hanging chains or ropes, Cartesian ovals for optics, etc. There are many interesting algebraic curves that appear in illustrations of works on geometry but are as rare as hen's teeth in technical drawings.

Among the plane figures consisting of straight lines, triangles, squares and rectangles are the most common, while other regular polygons are occasionally used.

Because of their importance in technical drawings and because the subject particularly interests me, in addition to **path**, 3DLDF defines the following object types.

For representing polygons:

- **triangle**
- **rectangle**
- **polygon**
- **reg_polygon** (regular polygon)

For the conic sections, 3DLDF implements the following types:

- **circle**
- **ellipse**
- **parabola**
- **hyperbola**

In addition, and unlike METAFONT, 3DLDF defines **superellipse** as an object type.

3DLDF also has types for solid geometric figures. Figures consisting of straight lines:

- **cuboid**
- **polyhedron**

Figures with a simply curved surface:

- **cone**
- **cylinder**

Quadric surfaces:

- **sphere**
- **ellipsoid**
- **paraboloid**
- **hyperboloid**

In addition, 3DLDF implements many predefined constants of these types:

```
unit_square  unit_pentagon unit_circle
unit_ellipse unit_cuboid   unit_sphere
unit_ellipsoid ...
```

The planar figures `unit_ellipse`, etc., are constructed in the x-z plane:

```
beginfig(2);
  ellipse e;
  e := unit_ellipse scaled (2cm, 0, 1cm);
  draw e;
endfig with_projection parallel_x_z;
```

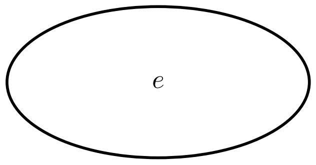


Fig. 3.

Ultimately, the solid figures consist of planar **path**s, a **polyhedron** of **polygon**s, a **sphere** of **circle**s, an **ellipsoid** of **ellipse**s, etc., and additional information is stored for the center, foci, axis lengths or other salient features of the figure.

They may be used in technical drawings for *wire-frame* constructions. 3DLDF doesn't implement any form of surface hiding or rendering. In the case of cuboids or polyhedra, surface hiding may be done "by hand". For drawings that aren't too complex, this may work well enough:

```
cuboid c[];
rectangle r[];
c0 := unit_cuboid scaled (1.5, 1.5, 1.5);
c1 := c0;
c2 := c0;
rotate c1 (0, 30);
shift c1  (-1.5cm, 0);
draw c1;
for i = 0 upto 5:
  r[i] := get_rectangle(i) c1;
endfor;
rotate c2 (0, -30);
shift c2 (1.5cm, 0);
draw c2;
for i = 0 upto 5:
  r[i+6] := get_rectangle(i) c2;
endfor;
unfilldraw r6;
unfilldraw r9;
unfilldraw r10;
```

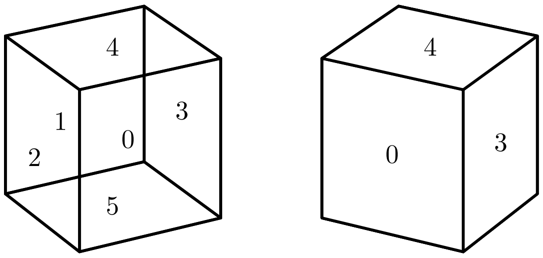


Fig. 4.

For curved surfaces, such as spheres or ellipsoids, this isn't possible, as finding the curve that represents the edge of a curved surface from a particular point of view is non-trivial. Please notice the left and right edges where the nearly horizontal circles appear to extend slightly past the nearly vertical ones.

```
sphere s;
s := unit_sphere scaled (2.5, 2.5, 2.5);
rotate s (10, 11, 10);
draw s;
```

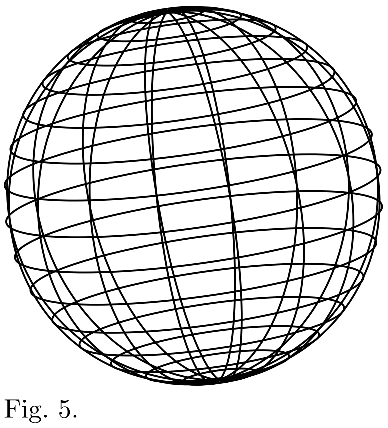


Fig. 5.

In the long run, the best solution for this problem would be to have 3DLDF produce output in a form suitable as input for a specialized rendering program, such as Blender (`https://blender.org`).

Path details. In METAFONT, **path**s are implemented as Bézier curves, which are a kind of *spline curve*. They may be specified in various ways, but are ultimately stored in the form [7, p. 13]:

$p = z_0$ .. controls u_0 and v_1 .. z_1 ⟨*etc.*⟩
z_{n-1} controls u_{n-1} and v_n .. z_n

As in this example:

```
tracingonline := 1; % METAFONT
path p;
p = fullcircle xscaled 2 yscaled 3;
show p;
>> Path at line 4:
(1,0)..controls (1,0.39784)
   and (0.89465,0.77939)
 ..(0.70712,1.06068)..
 controls (0.51959,1.34198)
 and (0.26523,1.5) ..(0,1.5)
```

Bézier curves are *invariant* under the affine transformations translation (shifting), rotation, and scaling, as long as they are scaled by the same amount in all dimensions [7, p. 132].

Unfortunately, Bézier curves are *not* invariant under the non-affine perspective projection, which is essential for 3D graphics. There is a generalization of the Bézier curve, that is, another spline curve, that is invariant under this projection, namely non-uniform rational B-splines or NURBs. With Bézier curves, conceptually, the control points exert a "pull" on the path and the amount of "pull" is equal for all of the control points. NURBs, on the other hand, have an additional "weight" parameter which makes it possible to "weight" the control points individually, so that they can exert different amounts of "pull". A Bézier curve is therefore equivalent to a similar NURB where all of the weights are 1.

One of the most important features of METAFONT is that, given a set of points on a **path**, it will attempt to draw the "most pleasing" curve through it [7, ch. 3, "Curves", p. 13ff.]. It provides a set of operations for the user to influence the shape of the curve by giving "hints": **dir**, **tension**, **atleast**, etc.

NURBs are not yet implemented in 3DLDF, although this is planned. **path**s are stored as the **point**s on the **path** and *connectors*, also known as *path joins*, may be specified in the same way as in METAFONT, i.e., with control points, **tension**, **dir**, {**point**}, $\{(x, y)\}$, $\{(x, y, z)\}$, and **atleast**. They are *not* converted to the form using "controls", as above, within 3DLDF but rather the connectors are written verbatim to the output (METAPOST and METAFONT code), except that any **point**s referred to are transformed along with the **path** whenever a transformation, including the projection upon output, is applied to the **path**. They therefore will only produce correct results for 2D objects that lie in the plane of projection, or a plane parallel to it, when a parallel projection is being used.

When METAFONT or METAPOST is run on the 3DLDF output, it will attempt to draw the "most pleasing" curve through the points on the path, accounting for any hints given with **dir**, **tension**, etc. *However*, METAFONT and METAPOST's idea of the "most pleasing" curve is based purely on the two-dimensional points on the path: They don't "know" that they represent the projection of a three-dimensional curve and the results are likely to be neither pleasing nor correct, *unless* the path is constrained by containing a sufficient number of points. In the special case that a parallel projection is used and the path lies in the plane of projection or a plane parallel to it, the result will, however, be correct, as it will be equivalent to just using METAFONT or METAPOST in the first place.

```
circle c;
set c with_center origin with_diameter 4
   with_point_count 8;
rotate c (-10, 0);
draw c;
```

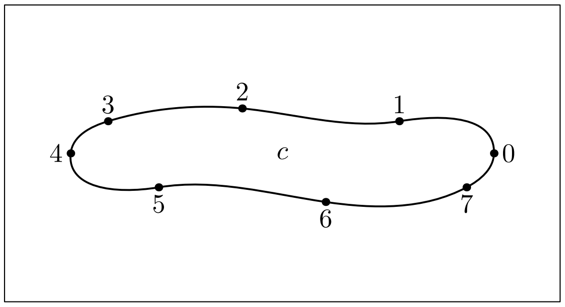


Fig. 6.

```
circle c;  % same, but with more points
set c with_center origin with_diameter 4
   with_point_count 64;
rotate c (-10, 0);
draw c;
```

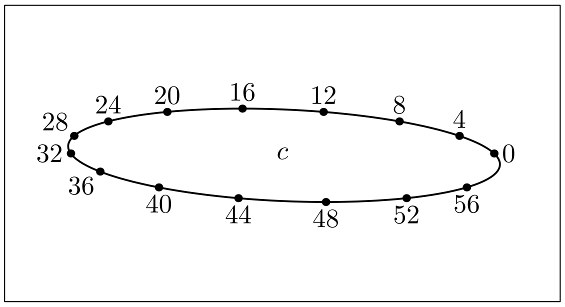


Fig. 7.

When projecting a 3D path using the perspective projection, **tension**, **dir**, $\{(x, y)\}$ and **atleast** are very likely to produce erroneous results, especially if an insufficient number of points have been specified. If the path is constrained sufficiently, this may not be noticeable; however, if any of these hints are used, the best practice is to replace the connectors with `..` before outputting the path, e.g.:

```
path p;
p := origin{up} .. {right}(1, 1){left}
     .. {down}(2, 0);
clear_connectors p;
p += ..;
```

Usually, there is no need to use hints, unless it is intended to use a parallel projection; nevertheless, the situation might arise, especially when working with METAFONT or METAPOST code "borrowed" from another source, such as the original sources for Computer Modern or any PostScript font accessed by means of the **glyph** command.

The situation is somewhat different with control points. Control points may be specified by the user and paths obtained by calling METAPOST from within 3DLDF will always contain them (unless they are subsequently removed).

Intersections. In METAFONT, since all **path**s are constructed in the same way, intersections are found by a method particular to Bézier curves (not described in *The METAFONTbook*). Furthermore, since METAFONT doesn't "know" that two **path**s p_0 -- p_1 and p_2 -- p_3 represent straight lines, their intersection is only found if it lies on the line segments they represent, *not* if the lines that contain them intersect.

In contrast to METAFONT, 3DLDF does "know" that a **path** like $(0, 0)$ -- $(1, 1)$ represents a straight line, that a **circle** represents a circle, and so on, and this is of importance in functions that attempt to find the intersections of objects in a drawing.

In addition, 3DLDF *does* find the intersection point of two lines (within reason) rather than just the intersection points of the line segments. Not within reason would be two lines that are almost parallel so that their intersection point lies outside the region where the computer could calculate it reliably.

```
point p[];
path q[];
q0 := origin -- (2cm, 2cm);
q1 := (0, 2cm) -- (2cm, 0);
draw q0;
draw q1;
p0 := q0 intersectionpoint q1;
```

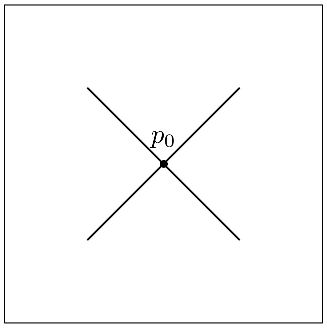


Fig. 8.

```
q0 := origin -- (2cm, 2cm);
q1 := (3cm, 0) -- (2.5cm, 2cm);
draw q0;
draw q1;
p0 := q0 intersectionpoint q1;
```

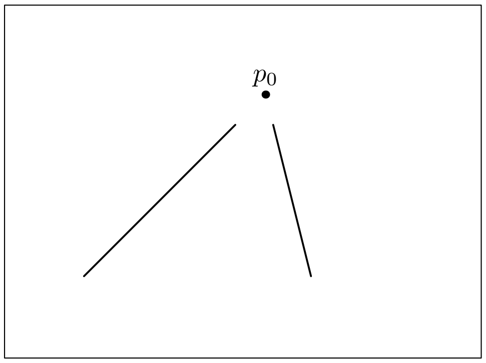


Fig. 9.

In this example, the result of **intersectionpoint** is stored in a **point**. In fact, the result is really an object of a *compound type* named **bool_point** (not present in METAFONT) consisting of a **boolean** and a **point**. In the case of the **intersectionpoint** operation, the **boolean** part of the **bool_point** indicates whether the **point** (if found) lies on one or both of the line segments.

If the lines do not intersect, the **point** returned in the **bool_point** is `INVALID_POINT` whose coordinates are the triple `(INVALID_REAL, INVALID_REAL, INVALID_REAL)`. Actually, neither `INVALID_POINT` nor `INVALID_REAL` are "invalid" from the point of view of the hardware or C++: `INVALID_REAL` is simply an arbitrary numerical value used within 3DLDF for testing whether an operation has succeeded or not. In fact, it is the largest **float** value available on a given computer.

The ability of 3DLDF to find the intersections of lines is useful as a way of compensating for its inability to interactively solve linear equations like METAFONT. In the latter, intersections are typically found using the "nullary" operation **whatever**, which is defined in `plain.mf` and `plain.mp` as a **vardef** macro [7, p. 264]:

```
path q;  % METAPOST
q0 = origin -- (2cm, 2cm);
q1 = (3cm, 0) -- (2.5cm, 2cm);
draw q0;
draw q1;
z0 = whateverorigin, (2cm, 2cm);
z0 = whatever(3cm, 0), (2.5cm, 2cm);
dotlabel.lft(btex $z_0$ etex, z0);
```

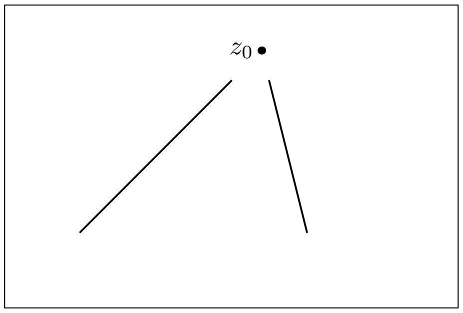


Fig. 10.

Intersections in 3DLDF. Finding the intersections of algebraic curves and surfaces in the plane and in 3D space is a focus of 3DLDF and a particular interest of mine. In some cases this is straightforward, whereas in others it is less so, or would involve higher mathematics currently beyond my abilities. Depending on the objects, their intersections may be points, curves, planes or curved surfaces.

The routines for finding intersections as in the previous examples all use the algebraic formulæ for the objects involved. One problem with this approach is that calculations involving real values (**float**s, **double**s or **long double**s) on a computer are not exact, so that there are always rounding errors. These in turn may cause intersections that exist to not be found. In particular, transforming objects by means of matrix multiplication tends to introduce rounding errors. For numerical a, b and ϵ, the solution is not to compare $a = b$, but rather $||a| - |b|| < \epsilon$ where ϵ is some small value appropriate to the circumstances.

Another problem is that transformations may cause objects to “go out of shape”. 3DLDF places no restrictions on the transformations that may be applied to a drawable object (**point**, **path**, **circle**, etc.). This feature would be useful, for example, when projecting a geometrical object like a circle onto a curved surface: The resulting figure will not be circular, but it retains the object type **circle** and its “center” will be the projection of the center of the original object onto the surface.

In order to deal with this problem, 3DLDF implements tests for whether objects of a given type still fulfill the definition of that type. For example, objects may be tested for circularity with the operator **is_circular**, for whether they are elliptical with **is_elliptical**, etc. In contrast, the operations **is_circle**, **is_ellipse**, etc., test for the object type:

```
beginfig(11);
circle c[];
c0 := unit_circle;
c1 := c0 sheared (1, 1.5, 1.25);
draw c0;
draw c1;
show is_circle c0;
>> true
show is_circle c1;
>> true
show is_circular c0;
>> true
show is_circular c1;
>> false
endfig with_projection parallel_x_z;
```

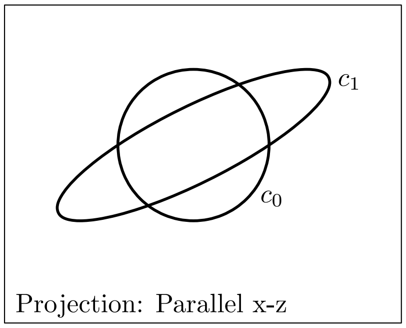


Fig. 11.

Vector-type objects

In METAFONT, *arrays* may be declared like this:

```
numeric n[];
path p[][];
```

Now, the user can assign values to the variables **n**⟨*suffix*⟩ and **p**⟨*suffix*⟩⟨*suffix*⟩:

```
tracingonline := 1; % METAFONT
n0 = 10;
n[n0 - 5] = 6;
show n0;
>> 10
show n[n0 - 5];
>> 6
a = 3;
p[a][n0] = origin .. (1, 1), -- (2, 0) .. cycle;
show p[3][10];
>> Path at line 10:
(0,0)..controls (-0.11444,0.59329)
   and (0.40671,1.11444)
 ..(1,1)..controls (1.33333,0.66667)
 and (1.66667,0.33333)
 ..(2,0)..controls (1.78766,-1.10071)
 and (0.21234,-1.10071) ..cycle
```

METAFONT makes it possible to declare variables using various combinations of *tags* and *suffixes* in a very flexible way [7, ch. 7, “Variables”, p. 53ff.]. This works in exactly the same way in 3DLDF. However, continuing the previous example, in METAFONT, a variable *n* would be completely independent of `n0`, `n[n0 - 5]`, etc.:

```
numeric n[];
*n0 := 10;
*n := 5;
*show n0;
>> 10
show n;
>> 5
```

To access all the variables of the form `n`⟨*suffix*⟩, they must be accessed individually. A loop may be used, but then the suffixes must follow a pattern suitable for use as a loop index:

```
for i = 0 upto 5:
   n[i] = 2i;
   show n[i];
endfor;
>> 0
>> 2
>> 4
>> 6
>> 8
>> 10
```

In order to make it more convenient to access all of the members of an array, 3DLDF implements the notion of *vector-types*. In this case, the term *vector* is used in the sense common in the context of computer programming, namely for one-dimensional arrays.

For most types, such as **boolean**, **transform**, **point**, **path**, etc., there is a corresponding vector-type: **boolean_vector**, **transform_vector**, **point_vector**, **path_vector**, etc. For some more rarely-used or specialized types, there is no corresponding vector-type.

There are operations that take vector-type objects as their arguments and operate on all of the objects on the vector. In addition, the operator `+=` may be implemented for a vector-type, where appropriate. For example, it adds a **path** to a **path_vector**:

```
path_vector pv;
pv += (1, 1) -- (2, 2);
pv += origin .. (1, 1, 1) .. (-1, 2, 2)
      .. (3.5, 0, 10);
show pv;
>> path_vector:
size of vector: 2
0:
type:  PATH_TYPE
surface_hiding_ctr:  0
decomposition_level:  0
points.size() == 2
connector_type_vector.size() == 1
points:
(1.00000000, 1.00000000, 0.00000000) --
   (2.00000000, 2.00000000, 0.00000000) ;
 etc.

1:
type:  PATH_TYPE
surface_hiding_ctr:  0
decomposition_level:  0
points.size() == 4
connector_type_vector.size() == 3
points:
(0.00000000, 0.00000000, 0.00000000)
   .. (1.00000000, 1.00000000, 1.00000000)
   .. (-1.00000000, 2.00000000, 2.00000000)
   .. (3.50000000, 0.00000000, 10.00000000) ;
 etc.
```

Vector-type variables can also be used as the return types for operations so that operations may return more than one object. For example, the operation ⟨*ellipse tertiary*⟩ **intersectionpoints** ⟨*ellipse secondary*⟩ returns 0 to 4 points:

```
ellipse e[];
e0 := unit_ellipse scaled (2cm, 0, 1cm);
e1 := e0 rotated (0, 45) shifted
   (.25cm, 0, -.125cm);
rotate e0 (0, -20);
draw e0;
draw e1;
point_vector P;
P := e0 intersection_points e1;
show size P;
>> 4
```

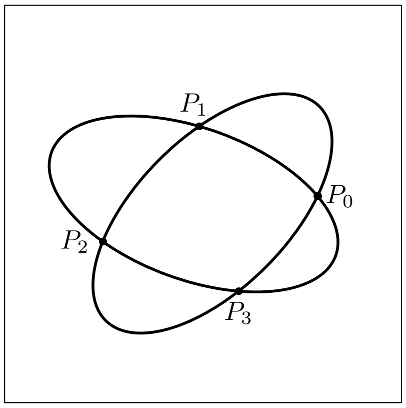


Fig. 12.

It is not possible to declare an array of vector-type objects.

Output and labels

In METAFONT, output is caused by the **shipout** primitive, which "ships out" a character. Normally, it is not called directly by the user, but rather indirectly by the **endchar** macro defined in `plain.mf`, which calls the macro **shipit**, which in turn calls **shipout** with *currentpicture* as its argument.

In METAPOST, **endfig** calls **shipout**, causing PostScript code to be written to the output file [4, p. 46].

In 3DLDF, **endfig** causes *current_picture* to be written to the METAPOST output and **endchar** causes it to be written to the METAFONT output. However, in addition, the **output** operator will cause the **picture** given as its argument, which may be **current_picture**, to be output in the form of METAPOST or METAFONT code to be written to the corresponding output file if called between **beginfig** and **endfig** on the one hand or **beginchar** and **endchar** on the other. This is useful for performing a primitive kind of surface hiding by hand and for creating figures or characters using more than one projection.

output doesn't clear the **picture** passed to it as an argument, so it may be output again, by **output**, **endfig** and/or **endchar**. If it should be cleared, it must be cleared explicitly using the **clear** command.

```
beginfig(1);
  draw unit_circle scaled (2cm, 0, 2cm);
  output current_picture with_projection
     parallel_x_z;
  clear current_picture;
  draw unit_ellipse scaled (3cm, 0, 2cm)
     rotated (90, 0);
   (etc.)
endfig with_focus f;
```

In METAFONT, labels are only used when developing a font and disappear in the final output. The placement is also usually determined by METAFONT and not by the user. They are therefore less important than in METAPOST and 3DLDF, since labels are an essential part of technical drawings.

In 3DLDF, labels work as in METAPOST, with only a few differences. METAPOST typesets labels by default using TeX. The `-tex` option may be used to set the name of the program to be called, e.g., `-tex=latex`.

In METAPOST, a plain **string** is typeset in the default font. On my system, this is Computer Modern Roman 10pt (`cmr10`), but it could be any PostScript font. To use TeX macros in the labels, **btex** ... **etex** or the macro **TEX** must be used. If METAPOST is called with the `-troff` argument, then `troff` is used to typeset material surrounded by **btex** and **etex** (or **verbatimtex** and **etex**). The first argument to **label** or **dotlabel** may be a **string** or a **picture**. See [4, ch. 8, "Integrating Text and Graphics"] for more information.

3DLDF passes the text in labels to METAPOST largely unchanged, so how they are handled depends on how METAPOST is called by the user after 3DLDF has run. However, **btex** and **etex** are always added to the beginning and end, respectively, of the label text. If any user found this undesirable, it would be easy to add an option to suppress this. A **picture** is not permitted as the first argument to **label** or **dotlabel**, but a number may be used, e.g., `dotlabel(0, p0)`.

In 3DLDF, a **picture** contains two kinds of items: 1) *drawable* ones, namely **point**s, **path**s, objects derived from **path** such as **circle** and **ellipse** and **solid**s, such as **sphere** and **ellipsoid** and 2) labels. When a **picture** is output, the drawable items are output first, followed by the labels, so that the latter aren't overwritten. The latter effect can be achieved by use of the **output** command to output a **picture** containing labels prior to **endfig**.

In 3DLDF programs, label commands may also appear between **beginchar** and **endchar**. When a **picture** is output, any labels on it are ignored. Thus the METAFONT output of 3DLDF will never contain labels, even when **mode** is **proofing**. To test the appearance of a character, it's best to produce a corresponding METAPOST figure, where one has the advantage of METAPOST's superior labelling facilities.

Labels in 3DLDF are purely two-dimensional items. Conceptually, they always lie in the plane of projection. The **point** specified as the second argument of **label** or **dotlabel** is projected like all of the other points on the picture and the label is placed in relation to its projection. The **point** may be transformed but the label itself may also be transformed separately. If rotation is desired, it should be rotated about the z-axis. Shifting and rotating make sense; other translations are likely to produce undesirable results.

Calling METAPOST from within 3DLDF

Having numerous object types to store information about algebraic curves and surfaces has certain advantages compared to storing them simply as spline curves, e.g., the ability to access their centers, foci, normals, radii, etc. On the other hand, it's nice to be able to find the intersections of arbitrary curves and surfaces, even free-form ones. Since 3DLDF does not yet implement NURBs, this is unfortunately not possible for the general case of intersections of arbitrary objects in 3D. However, for the special case of coplanar objects, it *is* possible by means of calling METAPOST (not METAFONT!) "indirectly" from within 3DLDF.

METAPOST is called instead of METAFONT because in METAPOST, the type used for arithmetical calculations may be specified using the option `-numbersystem`, allowing for almost arbitrary precision. In METAFONT, calculations are performed using 32-bit fixed-point numbers, with a strict limit on the magnitude of a ⟨*numerical token*⟩, namely < 4096 [7, pp. 50 and 63].

METAFONT implements many operations on **path**s, as described in *The METAFONTbook*. Because 3DLDF only stores the **point**s on a **path** and the connectors (or *path joins*), and does not calculate the "in-between" points, it is not possible to implement any of the operations that depend on this data. However, in the case of planar **path**s, it is possible to access these operations by calling METAPOST from within 3DLDF.

The operations in question are the following, where n stands for a numerical value, t for a numerical "time" value, p for a **pair** and q for a **path**:

- $p =$ **direction** t **of** q;
- $t =$ **directiontime** p **of** q;
- $p_0 =$ **directionpoint** p_1 **of** q;
- $p =$ **point** t **of** q;
- $p =$ **subpath** (t_0, t_1) **of** q;
- $p = q_0$ **intersectiontimes** q_1;
- $p = q_0$ **intersectionpoint** q_1;

The operation **angle** on a **pair** can be useful in combination with operations on **path**s:

- $n =$ **angle** p;
- $n =$ **angle direction** t **of** p;

Some of these operations have correspondences in 3DLDF for arbitrary (3D) **path**s, but they don't necessarily have exactly the same meaning or work the same way. In some cases, the names of the 3DLDF operations for operating on planar paths by calling METAPOST differ from the names of the corresponding operations in METAFONT names in order to avoid name clashes.

In the following listing, m and u are integers, v is a **numeric_vector** and w is a **point_vector**.

- q_0 := **get_metapost_path** q_1 $\langle options\rangle$
- v := **direction_metapost of** q $\langle options\rangle$
- n := **angle_direction_metapost** n **of** q $\langle options\rangle$
- v := q **intersectiontimes** q $\langle options\rangle$
- v := q **intersectiontimes_all** q $\langle options\rangle$
- p := q **intersectionpoint_metapost** q $\langle options\rangle$
- w := q **intersectionpoints_metapost** q $\langle options\rangle$
- q_0 := **resolve** q_1 (m, n) **to** u $\langle options\rangle$
- q_1 := **subpath** (n, n) **of** q
- q_0 := q_1 **normalized** $\langle options\rangle$

Commands:

- **call_metapost** $\langle$*string expression*$\rangle$ **(** $\langle$*path vector variable optional*$\rangle$ $\langle$*point vector variable optional*$\rangle$ $\langle$*numeric vector variable optional*$\rangle$ **)** $\langle$*options*$\rangle$
- **call_tex** $\langle$*string expression*$\rangle$ **,** $\langle$*numeric vector variable*$\rangle$ $\langle$*save optional*$\rangle$
- **normalize** $\langle$*path variable*$\rangle$ $\langle$*options*$\rangle$

In order for this to work, the 3DLDF path must first be put into the x-y plane. Then, a corresponding METAPOST path is created, the desired operations are performed on it, and the generated data is returned. In the case of points, they must be transformed by the inverse of the transformation that placed the path into the x-y plane.

```
circle c[];
transform t[];
c0 := unit_circle scaled (1cm, 0, 1cm)
   rotated (90, 0);
t0 := (identity rotated (0, 60, 20))
   shifted (1.9cm, 1cm);
c0 *= t0;
c1 := c0 normalized save;
draw c0;
draw c1;
```

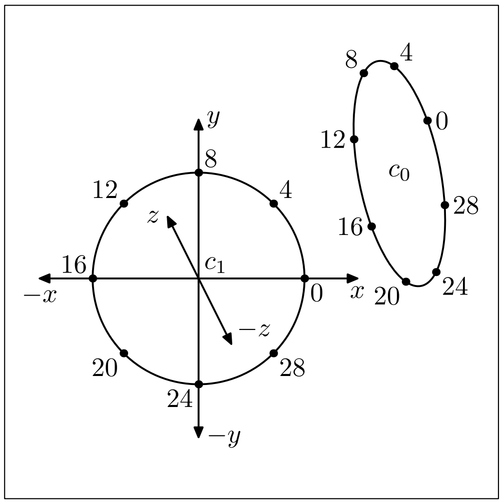


Fig. 13.

```
ellipse e[];
transform t[];
e0 := unit_ellipse scaled (1.75cm, 0, 1.25cm)
   rotated (90, 0);
t0 := (identity rotated (0, 60, 20))
   shifted (1.9cm, 1cm);
e0 *= t0;
e1 := e0 normalized save;
draw e0;
draw e1 with_color dark_gray;
```

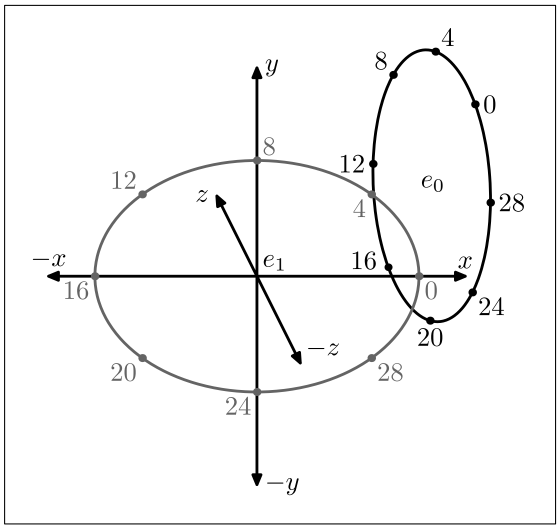


Fig. 14.

Further information and getting help

This article is merely intended to provide an introduction to GNU 3DLDF and to make it possible for the reader to make a start on using it, if desired. 3DLDF is a large and complex program and to cover its use and the ideas behind it comprehensively would require considerably more space. Many topics touched upon here could be enlarged upon in subsequent articles.

Unless one already knows METAFONT or METAPOST, the best way to learn 3DLDF is to start with *METAPOST, A User's Manual* [4]. For a deeper understanding of METAFONT, METAPOST and 3DLDF, *The METAFONTbook* is indispensable. After this, for learning 3DLDF itself, the GNU 3DLDF website provides further information and many examples of code and the generated graphics:
`https://www.gnu.org/software/3dldf`.

The mailing list `help-3dldf@gnu.org` is available for users for discussion or to ask for help. See `https://www.gnu.org/software/3dldf/#Mailing_lists` for instructions on how to subscribe.

Unfortunately, the last edition of *The 3DLDF User and Reference Manual* (`https://www.gnu.org/software/3dldf/#Documentation`) is from 2006 and sadly out-of-date. At that time, 3DLDF was not yet interactive, as I had not yet implemented the scanning and parsing routines. It was more of a software library and drawings had to be created by writing C++ code to be linked to the latter. While it provides a good starting point for anyone who wants to know how 3DLDF works internally, it, of course, doesn't document the many features I've added since 2006.

Links

The GNU 3DLDF website:
`https://www.gnu.org/software/3dldf`

The author's personal website:
`https://laurence-finston.de`

METAPOST on the Web:
`https://tug.org/metapost.html`

CTAN, METAFONT package page:
`https://www.ctan.org/pkg/metafont`

Bibliography

[1] Cundy, H. Martyn and A.P. Rollet. *Mathematical Models.* Oxford: Oxford University Press, 1961.

[2] Gardner, Martin. *Mathematical Carnival.* New York: The Mathematical Association of America, 1989.

[3] Gonçalves, L. N. *FEATPOST and a Review of 3D METAPOST Packages.* In *TEX, XML, and Digital Typography. TUG 2004.* Lecture Notes in Computer Science, vol. 3130, p. 112ff. Berlin, Heidelberg: Springer, 2004. `https://doi.org/10.1007/978-3-540-27773-6_8`

[4] Hobby, John D. and the MetaPost development team. *METAPOST, A User's Manual.* 2020. `https://tug.org/metapost`

[5] Jones, Huw. *Computer Graphics through Key Mathematics.* London, UK: Springer-Verlag Limited, 2001.

[6] Knuth, Donald E. *The Art of Computer Programming.* Boston: Addison Wesley Publishing Company, 2019.

[7] Knuth, Donald E. *The METAFONTbook.* Reading, Massachusetts: Addison Wesley Publishing Company, 1986.

[8] Knuth, Donald E. *The TEXbook.* Reading, Massachusetts: Addison Wesley Publishing Company, 1986.

[9] Knuth, Donald E. *METAFONT: The Program. Computers & Typesetting*, vol. D. Reading, Massachusetts: Addison Wesley Publishing Company, 1986.

[10] Ramsey, Norman. *A Simple Solver for Linear Equations Containing Nonlinear Operators.* `https://www.cs.tufts.edu/~nr/noweb/examples/solver.html`

[11] Wikipedia. *Non-uniform rational B-spline.* `https://en.wikipedia.org/wiki/Non-uniform_rational_B-spline`

◇ Laurence Finston
Germany
`Laurence.Finston@gmx.de`

A graphical ellipse envelope construction with GNU 3DLDF

Laurence Finston

Abstract

This article demonstrates the use of GNU 3DLDF for a graphical solution of the problem of constructing the envelope of an ellipse and an approximation to the curve itself.

Introduction

Before the universal availability of computers and graphics software, technical drawings had to be made by hand by draftsmen and -women, who were skilled professionals. Making a technical drawing of even moderate complexity was time-consuming, painstaking and error-prone work, requiring much knowledge and patience and the ability to endure frustration. Erasures were difficult and a single error of conception or execution could render useless the work of many hours.

Well into the 20th century, graphical methods for constructing curves were of importance for the creation of technical drawings. In addition, such constructions were of central importance in the mathematics of the ancient Greeks, especially those constructions that only made use of a straight edge and dividers, with the added restriction that the use of the dividers for measuring lengths was forbidden.

Even today, with computers and 3D graphic software, graphical methods of constructing geometric figures retain their fascination and continue to provide insight and diversion to those who appreciate elegant and ingenious solutions to mathematical puzzles. In fact, the use of computers greatly increases the pleasure of creating technical drawings, due to their speed, accuracy and the ease of making corrections.

This article demonstrates the use of GNU 3DLDF for a graphical solution of the problem of constructing an ellipse found in Lockwood's *A Book of Curves*, p. 13 [2]. GNU 3DLDF is a package for three-dimensional drawing with METAPOST and METAFONT output. It implements a language based on the METAFONT language with many additional data types and operations. More information can be found on the GNU 3DLDF website:
`https://www.gnu.org/software/3dldf/LDF.html`

The following figures are two-dimensional, so it would have been possible to create them using METAPOST alone. They do use several features of 3DLDF that are not present in METAPOST, but it could easily be adapted. For example, rotation about the z-axis could be replaced by calls to **reflectedabout**.

Constructing an ellipse envelope

Draw a circle c with center C (fig. 1). $\overline{AA'}$ is a diameter of c and S a point on $\overline{AA'}$. The distance CS should be $\geq \frac{3}{5}CA$. Q_5, Q_{10} and Q_{20} are points on the perimeter of c such that $\angle A'SQ_5 = 25°$, $\angle A'SQ_{10} = 50°$ and $\angle A'SQ_{20} = 100°$. R_5, R_{10} and R_{20} are points on the perimeter of c such that $\angle SQ_5R_5 = \angle SQ_{10}R_{10} = \angle SQ_{20}R_{20} = 90°$. S' is S rotated around C by 180°. That is, if $x_S = -k$, $x'_S = k$. S and S' are the foci of the ellipse.

With fixed S, as points Q_x, R_x for $1 \leq x \leq N-1$, $N = 72$ are added, whereby Q_x and R_x are on the same side of $\overline{AA'}$, their intersections will form an *envelope* describing an ellipse e with foci S and S' and major axis $\overline{AA'}$.

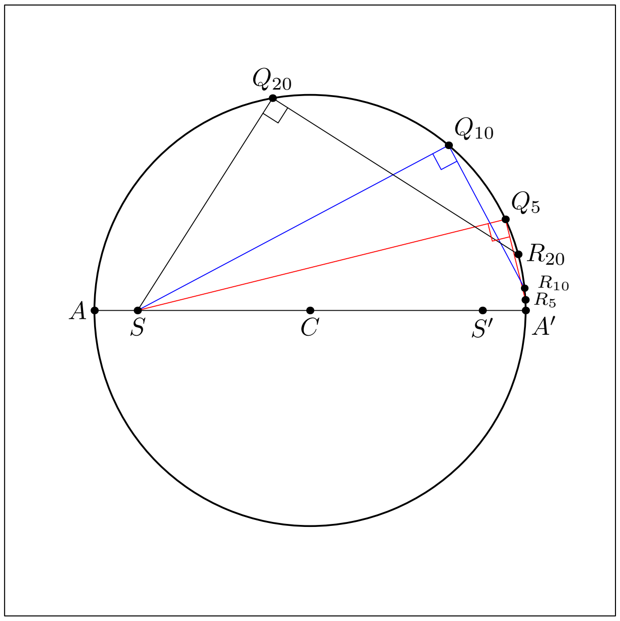


Fig. 1.

As points Q_x and R_x are added, it becomes clear that the intersections of the lines $\overline{Q_xR_x}$ quickly form an *envelope* revealing the outline of the ellipse. (see fig. 2).

If this figure were to be drawn by hand, two lines would have to be drawn for each of the Q_xR_x pairs and the right angles $\angle SQ_xR_x$ would have to be obtained. Placing a set square accurately so that Q_x and R_x were both squarely on the perimeter thirty times would be quite a challenge.

doi.org/10.47397/tb/43-3/tb135finston-ellipse

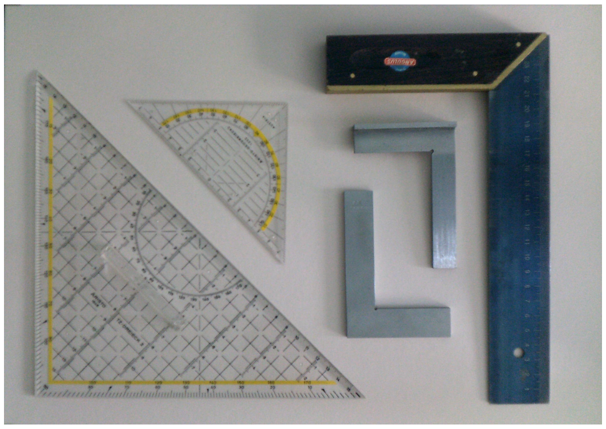


Plate 1. A selection of set squares.

Assuming this feat was accomplished, the next step would be to trace the curve of the ellipse. This could be done with a flexible curve or a French curve. I personally have never had good results with either of these tools, especially where the curvature was small.

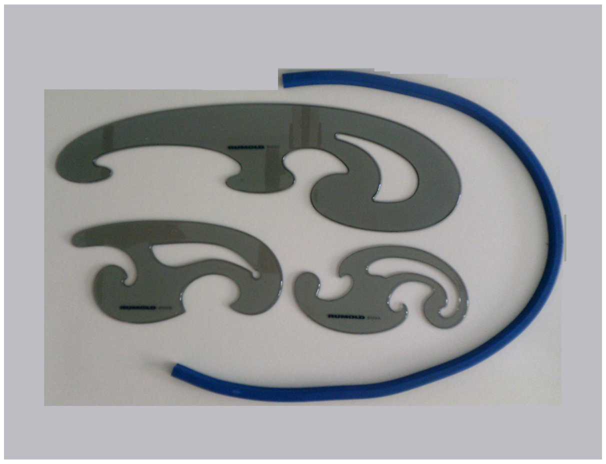


Plate 2. A set of French curves and a flexible curve.

On the other hand, when tracing a curve approximating the ellipse using points on the envelope using a computer, it doesn't suffice to intuitively recognize the rough shape of the ellipse. While it is not necessary to find all of the intersection points that are closest to the ellipse, it is necessary to find a sufficient number of them to trace a good approximation to the latter and to ensure that all of the points chosen are as close as possible to the ellipse.

Figure 2 shows the intersection points $p_{29} = \overline{Q_{29}R_{29}} \bigcap \overline{Q_{28}R_{28}}$ and $p_{30} = \overline{Q_{30}R_{30}} \bigcap \overline{Q_{29}R_{29}}$. The intersection points used are thus $p_x = \overline{Q_xR_x} \bigcap \overline{Q_{x-1}R_{x-1}}$ for $x > 1$.

In figure 2, p_{29} lies in the first quadrant of the ellipse, while $x_{p_{30}}$ lies in the second. It is not strictly speaking necessary to find the intersection points p_x for $x > 29$, since the intersection points in the first quadrant may simply be rotated about the x- and y-axes in order to obtain points close to the ellipse in the remaining three quadrants of e. Unless by chance, they will not, however, be the same points that would be found by continuing to find the intersection points of the lines $\overline{Q_xR_x}$.

By hand, this would be no less work than finding $p_1 \ldots p_{29}$ in the first place, but with the computer it is the work of a moment.

Q_{35} is already very close to A and $\overline{Q_{35}R_{35}}$ is not too far from being vertical. Q_{36}, in fact, coincides with A, $\overline{Q_{36}R_{36}}$ is vertical and the intersection point $p_{36} = \overline{Q_{36}R_{36}} \bigcap \overline{Q_{35}R_{35}}$ doesn't exist.

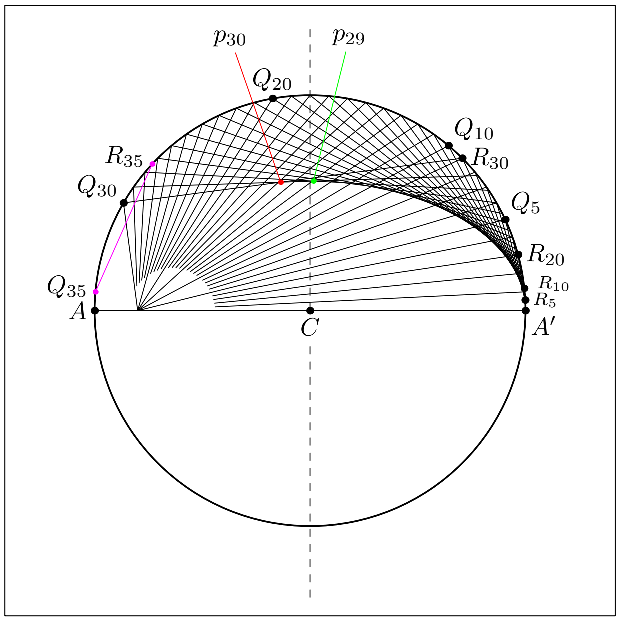


Fig. 2.

Figure 3 shows the result of continuing to find Q_x and R_x, up to Q_{71} and R_{71} (Q_{72}, or generally $Q_N = A$) and draw the lines $\overline{Q_xR_x}$. In this figure, some portions of the lines that converge at S have been erased so that the dots and labels may be seen and to avoid a large, unsightly splotch of ink around S.

S' is the reflection of S about the y-axis. S and S' are the foci of the ellipse. The major axis is $\overline{AA'}$ but the minor axis is not so easy to determine. It is twice the distance from C to a point w on the ellipse on the line through C perpendicular to $\overline{AA'}$, to the left of p_{29} and to the right of p_{30}, which are both close to the ellipse, but not actually on it. I will have to give some thought to how to determine w without “cheating”.

To execute this drawing in pen-and-ink would be a nightmare.

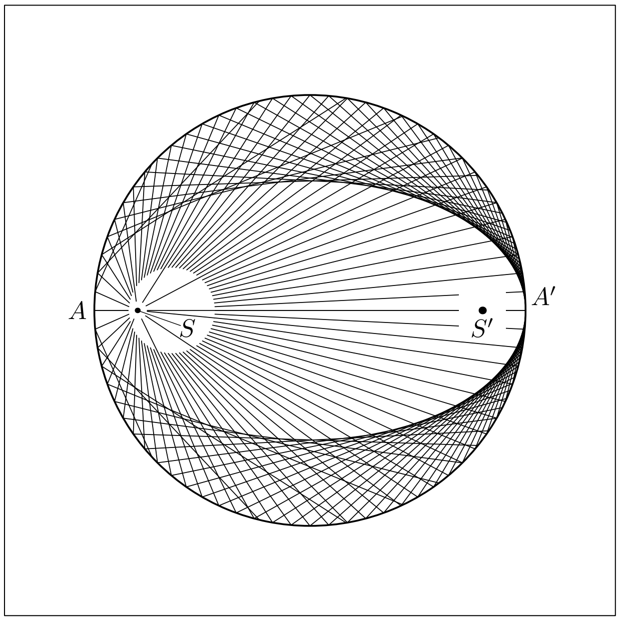


Fig. 3.

Figure 4 shows the intersection points $p_{10}, p_{15}, \ldots p_{70}$. Clearly, the length of arc $p_{x-1}p_x$ increases as p_x approaches A and decreases again as it passes it and approaches A' again.

The N points A, A' and p_x for $0 < x < N$, $N = 72$, $x \neq 36$ would normally be sufficient for creating an ellipse object in 3DLDF, if we were to consider them to be close enough to the ellipse to be usable, unless it were to be projected with extreme foreshortening, which requires there to be enough points on a path to prevent it from “going out of shape” when the projected path is passed to METAPOST for displaying or printing, as explained in the article “An Introduction to GNU 3DLDF” (pp. 319–332 in this issue).

It would nevertheless be somewhat unsatisfying to have such different arc lengths depending on the position on the circle of the points used for generating the envelope and using points that were only close to the ellipse inside of actually on it for the path. Instead, 3DLDF uses the parametric equation for an ellipse to generate the path for objects of type **ellipse**, i.e.,

$$(x, y) = (a \cos(t), b \sin(t)) \text{ for } 0 \leq t \leq 2\pi\,.$$

Using the intersection points $p_x = \overline{Q_xR_x} \bigcap \overline{Q_{x-1}R_{x-1}}$ appears to produce nearly correct results. It would seem that the intersection point of a line $\overline{Q_aR_a}$ with its adjacent lines $\overline{Q_{a-1}R_{a-1}}$ and $\overline{Q_{a+1}R_{a+1}}$ are closer to C than its intersection points with other lines $\overline{Q_xR_x}$ and would hence produce the closest approximation to an ellipse. However, I would have to think about whether I could prove this with my limited mathematical skills.

As examples of the positions of other intersection points, g_{43} is the intersection point $\overline{Q_{10}R_{10}} \bigcap \overline{Q_{25}R_{25}}$ and lies close to the perimeter of the ellipse and g_{44} is the intersection point $\overline{Q_5R_5} \bigcap \overline{Q_{30}R_{30}}$ and lies outside c.

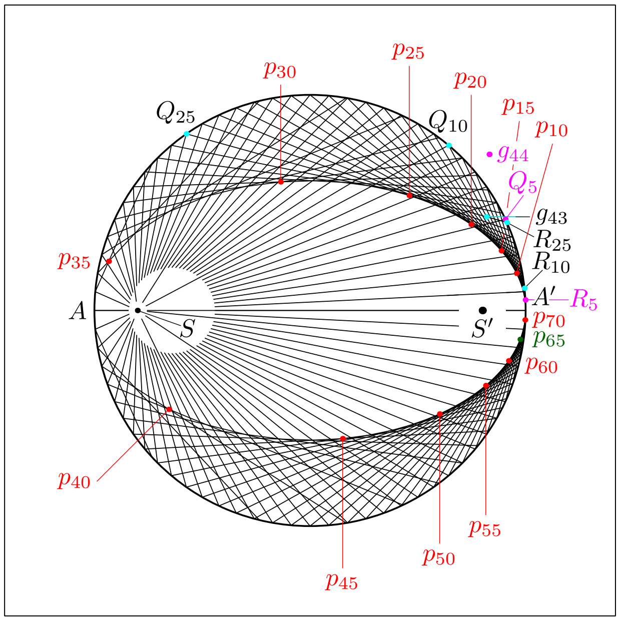


Fig. 4.

Figure 5 shows all of the intersection points $p_1 \ldots p_{35}$, $p_{37} \ldots p_{71}$, plus A and A', with A' and the points with odd indices in red and A and the ones with even indices in blue.

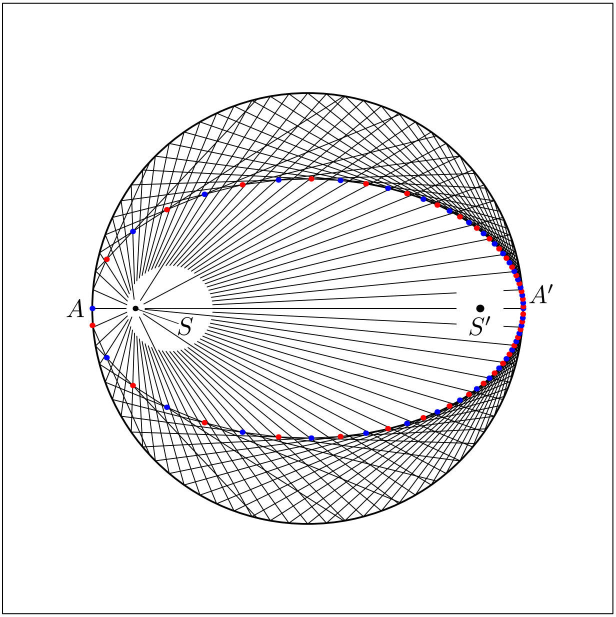


Fig. 5.

Figure 6 shows the quarter ellipse q_0 containing points A and the intersection points $p_1 \ldots p_{29}$. This figure also shows that $\overline{Q_{29}R_{29}}$ and $\overline{Q_{30}R_{30}}$ intersect at p_{30}.

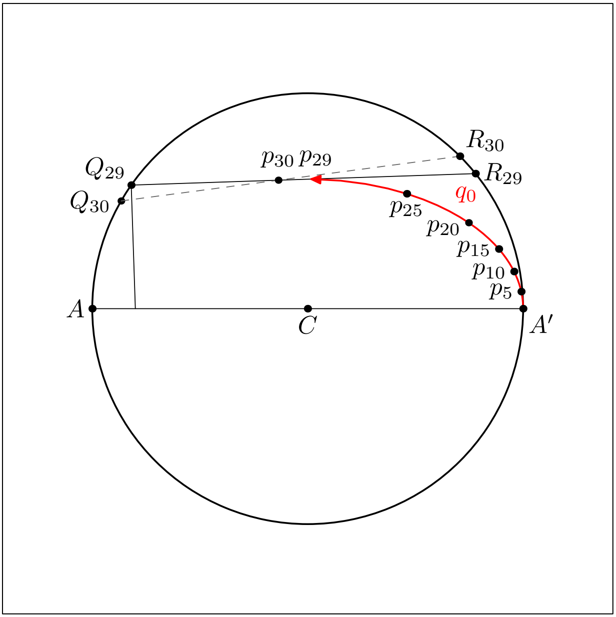


Fig. 6.

Figure 7 shows the completed approximation to an ellipse e_a consisting of the combination of the paths q_0, ${q_1}'$ and ${q_3}'$ where ${q_1}'$ is the reflection of q_0 about the y-axis with the order of the points reversed and ${q_3}'$ is $\overline{q_0{q_1}'}$ reflected about the x-axis and with the order of the points reversed.

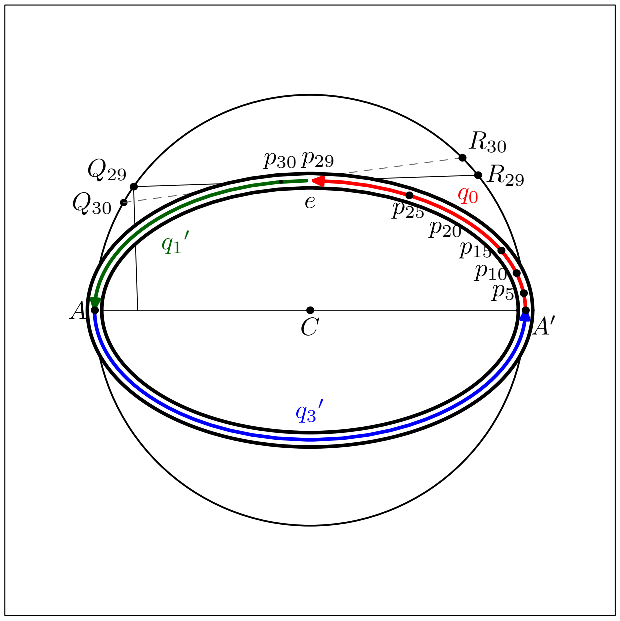


Fig. 7.

The annotated GNU 3DLDF code

The following listings contain only the parts of the 3DLDF program for the figures in this article that are of particular interest. Labels, “bookkeeping chores” and other items have been left out in the interest of comprehensibility. The full code may be found here: `https://www.gnu.org/software/3dldf/ellipses.html#Constructions`

Let’s start with some basic declarations:

```
point p[];
point R[];
point Q[];
point a[];
point d[];
path q[];
circle c[];
numeric n[];
boolean b[];
bool_point_vector bpv;
picture v[];
```

Everything here is just the same as it would be in METAFONT except for **point** `p[]` and the other **point** array declarations, **circle** `c[]` and **bool_point_vector** `bpv`. **point** is the 3D equivalent of **pair** and **circle** is a type in 3DLDF similar to a **path**, except its radius is stored as part of the object and there are special operations that apply to **circle**s that don’t apply to **paths**, such as **get_center** and the predicate **is_circular**.

Of course, as mentioned above, the figures in this article do not require any 3D calculations and could as easily have been created using METAPOST with a few changes. Nonetheless, in 3DLDF, points in space, whether two-dimensional or three-dimensional, are represented by objects of type **point** and the type **pair** doesn't exist.

bool_point is a type in 3DLDF that combines a boolean and a point in a single object. bool_point objects may be returned as the result of operations, such as intersection_point, whereby the boolean part indicates whether a particular condition is true or false, e.g., whether the point lies on one or both of the paths.

bool_point_vector is a 3DLDF type containing multiple bool_point objects. It is a so-called "vector-type". The latter are similar to arrays, e.g., bool_point[], except that a vector-type object may be returned as the result of an operation or operated upon as a single object, whereas these things aren't possible for arrays.

```
c0 := unit_circle scaled (3cm, 0, 3cm)
   rotated (90, 0);
draw c0;
point C;
C := origin;
point A;
A := get_point 16 c0;
point Aprime;
Aprime := get_point 0 c0;
draw A -- Aprime;
point S;
S := mediate(A, C, .2);
```

unit_circle is a predefined constant of type circle. Unlike METAFONT, where the "canonical" unit is the pixel and METAPOST, where it is the PostScript point or *big point* (1bp = 1/72 in), in 3DLDF, the canonical unit is the centimeter and thus unit_circle has radius (not diameter!) 1cm. And unlike fullcircle, which has 8 points in METAFONT, unit_circle has 32, thus point 16 of c_0 is at the halfway point around the circle.

32 points is normally about enough to prevent a **circle** from "going out of shape" when it is projected with a moderate amount of foreshortening. See "An Introduction to GNU 3DLDF" in this issue.

There are three other differences with respect to METAFONT in this section of the code:

- All of the assignments are actual assignments using := rather than equations using =. Unfortunately, 3DLDF doesn't (yet?) share METAFONT's "amazing ability to deduce explicit values from implicit statements" [1, p. 83].
- get_point is the equivalent in 3DLDF of point in METAFONT, as in ⟨pair primary⟩ ⟶ **point** ⟨numeric expression⟩ of ⟨path primary⟩ [1, p. 73].

  In 3DLDF, as previously mentioned, the symbol point is a "declarator" used to declare point objects and its use as an operator would have caused conflicts in the grammar generated by the parser generator GNU Bison.
- Nor did Bison allow METAFONT's syntax for the *mediation* operation, e.g., .5[p0, p1] because it would have conflicted with the other uses of brackets. Therefore, the operator mediate must be used instead.

In 3DLDF, as in METAFONT, A' would have been a valid name for a variable. However, I generally don't use such variable names and I thought it would be potentially confusing, so I used Aprime instead.

```
numeric j;
j := 0;
```

As in METAFONT, j could just have been used without explicitly declaring it as a numeric. Doing so is considered good programming style, although it may be overkill here and I have some doubts about whether every member of the "programming style police" actually has practical experience writing computer programs.

```
numeric N, k;
N := 72;
K := 360/N;
for i = 0 upto 100:
  Q[i] := Aprime rotated (0, 0, 0 + (i * K));
  d[j] := Q[i] shifted (0, 0, 1);
  d[j+1] := S rotatedaround (Q[i], d[j]) 90;
  bpv := (Q[i] -- d[j+1])
     intersection_points c0;
  a0 := bpv0;
  a1 := bpv1;
  if xpart a0 > xpart a1:
    a2 := a0;
  else:
    a2 := a1;
  fi
  R[i] := a2;
  j += 2;
endfor;
```

The loop in this section of the code is where the real action of the program begins. Q_x is found by rotating A' about C, the perpendicular to $\overline{SQ_x}$ through Q_x is found and R_x is found as the intersection point of the perpendicular with c_0 with the greatest x-coordinate. A circle and a line, that is, a path with two points and a simple connector, that is, one without any modifiers that would cause it to diverge from a straight line, can have 0 to 2 intersections.

Please note that this loop finds the points Q_x in a different way than in the description in section "Constructing an ellipse envelope" on page 333. However, it is completely arbitrary how these points are found.

Again, in most ways, the 3DLDF code would mostly be valid in METAFONT, but there are several important differences:

In METAFONT, rotation is about a 2D point, either the origin, with plain `rotated`, or about an arbitrary point with `rotatedaround`. In 3DLDF, rotation is about the x-, y- and z-axes with `rotated` and about an axis specified as two `points` with the operator `rotated_around`.

METAFONT provides the primitive operation `intersectiontimes` and a related macro named `intersectionpoint`, both of which return a single `pair` as their result. Therefore, if two `paths` intersect more than once, information about only one of the intersections is returned.

For 3DLDF, where geometric figures play a much greater role than in METAFONT, this behavior would not be acceptable, so `bool_point_vectors` are used as the type of the return values for the various operations that return intersection points or times.

The `a0 := bpv0` and `a1 := bpv1` statements show that `bool_points` can be assigned to `points`, whereby the `boolean` component of the `bool_point` is discarded. For many but not all operations in 3DLDF, `bool_points` may be used in place of `points`.

It is worth noting that the intersection points of lines with each other and lines with circles in this program are not found as in METAFONT. There, all `paths` are implemented as Bézier curves and intersection times and points are found with a routine that applies to all Bézier curves, irrespective of shape.

In 3DLDF, the intersections of lines with each other and with other geometric figures, such as circles, are used by combining and solving the implicit equations of the figures. However, since there are no restrictions on the transformations that can be applied to objects in 3DLDF, they must be tested to ensure that the equations still apply. For this purpose, 3DLDF implements the *predicate* operations `is_linear`, `is_circular`, `is_elliptical`, etc.

Here is the next portion of code we'll consider:

```
q0 += ..;
q1 += ..;
q0 += Aprime;
```

This, unlike what we've seen before, would not be valid METAFONT code. 3DLDF implements the operators `+=`, `-=`, `*=` and `/=` for the operations assignment with addition, subtraction, multiplication and division, respectively. While 3DLDF for the most part shares METAFONT's scanning rules, these operators break these rules, as `=` belongs to category 1, `+` and `-` to category 3 and `*` and `/` to category 4 [1, pp. 50–51]. However, this was easy to implement and has never caused any problems.

Here, the connector `..` is put onto `paths` `q0` and `q1` (which start out without any points) and `Aprime` is put onto `q0` as its first point.

```
for i = 1 upto (N - 1):
  if i <> 36:
    p[i] := (Q[i] -- R[i])
      intersection_point (Q[i-1] -- R[i-1]);
    b[i] := p[i] rotated (0, 180);
    if i < 30:
      q0 += p[i];
      q1 += b[i];
    fi
  else:
    message "Skipping p36.";
  fi
endfor;
v0 := current_picture;
```

A second loop finds the $N-2 = 70$ intersection points $p_x = \overline{Q_xR_x} \bigcap \overline{Q_{x-1}R_{x-1}}$ for $1 \leq x < N$, $x \neq 36$. $\overline{Q_{36}R_{36}}$ is skipped, because Q_{36} coincides with A and thus $\overline{Q_{36}R_{36}} \bigcap \overline{Q_{35}R_{35}}$ doesn't exist.

The intersection points are appended to `q0`. In addition, they are rotated 180° about the y-axis and appended to `q1`. Since ellipses are symmetrical about their major and minor axes, I only have to find the intersection points in the upper right quadrant of c_0 and can rotate them into the other quadrants instead of finding the intersection points in the latter, although that would certainly be possible with the construction described by Lockwood, whereby the points would be different.

Unlike the convention in TeX and METAFONT, where the names of macros, variables, etc., are run together, I favor the use of the underline character in variable names. However, `currentpicture` may be used as a synonym for `current_picture`, `rotatedaround` for `rotated_around`, `withpen` for `with_pen` and similarly for many other names of operators and predefined variables and constants.

Thus far, I have left out most of the drawing and labelling commands. However, the following are of special interest:

```
undrawdot Sprime with_pen pensquare
   scaled (.65cm, .65cm, .65cm);
dotlabel.bot("$S'$", Sprime);
undrawdot S with_pen pencircle
   scaled (.25cm, .25cm, .25cm);
drawdot S with_pen dot_pen;
```

About S and S': `undrawdot` uses a large circular or square pen, respectively, to clear out a space so that the labels may be seen and, the case of S, to avoid a large splotch of black ink. In addition, a selection of lines is drawn over the white space to show that the lines $\overline{Q_xR_x}$ converge at S.

```
q2 := q0 .. reversed q1;
q3 := q2 rotated (180, 0);
q4 := q2 .. reversed q3;
q4 += cycle;
drawarrow q0 with_color red
   with_pen medium_pen;
draw q0 .. reversed q1 .. reversed q3
        .. Aprime .. cycle
  with_pen pencircle
     scaled (2.5mm, 2.5mm, 2.5mm);
undraw q0 .. reversed q1 .. reversed q3
          .. Aprime .. cycle
  with_pen pencircle
     scaled (1.5mm, 1.5mm, 1.5mm);
drawarrow q0 with_color red;
drawarrow reversed q1 with_color dark_green;
drawarrow reversed q3 with_color blue;
```

This is the code that draws the constructed approximation to an ellipse e_a. I've included the drawing commands here because, together with the erasures above, this is a good example of technical drawing tasks that are trivial with the computer but would be difficult to execute by hand and likely to cause much wailing and gnashing of teeth.

To create the black outlines of e_a, it is first drawn using a large `pencircle` of 2.5mm diameter. To compare, in technical drawings, 0.5mm is the size used most. For example, acrylic templates for drawing shapes are designed for use with technical pens with 0.5mm nibs. (Other commonly available sizes are 0.25mm, 0.7mm and 1mm.) In this article, the "standard" pen size is 0.333mm. Then, the middle of the curve is cleared out by undrawing it with a `pencircle` of diameter 1.5mm. Finally, the paths q_0, ${q_1}'$ and ${q_3}'$, whereby the latter two are simply q_1 and q_3 reversed, are drawn in color and with arrows using a `pencircle` of diameter 0.5mm.

In METAFONT, `pencircle` would be scaled using a single numeric value while `pensquare` would be scaled using two. In 3DLDF, a drawing command copies an object such as a `path`, associates the copy with any items such as pens or colors that are specified in the command and puts them all together onto a `picture`, `current_picture` by default. The pens are only used when `endfig` or `output` causes METAFONT or METAPOST code to be written to an output file. They are therefore purely 2D objects. While they may be scaled using three numerical values, in fact only the x- and y-coordinates are used and the z-coordinate is ignored, even when the object is projected using the parallel projection onto the x-z plane.

Since this may change in the future, it is safest to specify all three dimensions when scaling a `pen`.

Acknowledgements. Many thanks to Denis Roegel and Bogusław Jackowski for improving this article with their corrections and suggestions.

References

[1] Knuth, Donald E. *The METAFONTbook.* Reading, Massachusetts: Addison Wesley Publishing Company, 1986.

[2] Lockwood, E.H. *A Book of Curves.* Cambridge, UK: Cambridge University Press, 1961.

◇ Laurence Finston
Germany
Laurence.Finston@gmx.de

Updates to “Automatically removing widows and orphans with lua-widow-control”, *TUGboat* 43:1 19012

infinite Max Chernoff

A request from *Zpravodaj*, the journal of the Czech/Slovak TEX group, to republish the subject article led to these updates. The section numbers here correspond to those in the original article. 2334

3.3 Clubs

In the original article, I discussed the origin of the typographical terms “widow”, “orphan”, and “club”. The first two terms are fairly well-known, but I had this to say regarding the third: 4078

> *The TEXbook* never refers to “orphans” as such; rather, it refers to them as “clubs”. This term is remarkably rare: I could only find a *single* source published before *The TEXbook* — a compilation article about the definition of “widow” — that mentions a “club line” [. . .] 39709
>
> I spent a few hours searching through Google Books and my university library catalogue, but I could not find a single additional source. If anyone has any more information on the definition of a “club line” or why Knuth chose to use this archaic Scottish term in TEX, please let me know! 1761

Conveniently, Don Knuth — the creator of TEX — read my plea and sent me this reply: 33108

> I cannot remember where I found the term “club line”. Evidently the books that I scoured in 1977 and 1978 had taught me only that an isolated line, caused by breaking between pages in the midst of a paragraph, was called a “widow”; hence TEX78 had only “`\chpar4`” to change the “`widowpenalty`”. Sometime between then and TEX82 I must have come across what appeared to be an authoritative source that distinguished between widows at the beginning of a paragraph and orphans or club lines at the end. I may have felt that the term “orphan” was somewhat pejorative, who knows? 1369

So this (somewhat) resolves the question of where the term “club” came from. infinite

9 Options

The overview to the “options” section stated that: 1092

> Plain TEX/OpTEX Some options are set by modifying a register, while others must be set manually using `\directlua`. 32617

However, this is no longer true. Now, commands are provided for all options in all formats, so you no longer need to use ugly `\directlua` commands in your documents. The old commands still work, although they will likely be removed at some point in the future. 17589

9.5 Penalties

`\brokenpenalty` now also exists as a LATEX and ConTEXt key. lua-widow-control will pick up on the values of `\widowpenalty`, `\clubpenalty`, and `\brokenpenalty` regardless of how you set them, so the use of these dedicated keys is entirely optional. 31907

9.6 `\nobreak` behaviour

The Plain/OpTEX command is now: infinite

`\lwcnobreak{`⟨*value*⟩`}` infinite

9.8 Draft mode

Since v2.2.0, lua-widow-control has a “draft mode” which shows how lua-widow-control processes pages. 1391

Plain TEX/OpTEX	`\lwcdraft 1`	1391
LATEX	`\lwcsetup{draft}`	1391
ConTEXt	`\setuplwc[draft=start]`	1391

Draft mode has been used for typesetting this article. It has two main features: 53221

First, it colours lines in the document according to their status. Any remaining widows and orphans will be coloured red, any expanded paragraphs will be coloured green, and any lines moved to the next page will be coloured blue. 3420

Second, this draft mode shows the paragraph costs at the end each paragraph, in the margin. 2659

This draft mode leads to a neat trick: if you don’t quite trust lua-widow-control, or you’re writing a document whose final version will need to be compilable by both pdfLATEX and LuaLATEX, you can load the package with: 16115

```
\usepackage[draft, disable]          infinite
          {lua-widow-control}        infinite
```

This way, all the widows and orphans will be coloured red and listed in your log file. When you go through the document to try and manually remove the widows and orphans — whether through the `\looseness` trick or by rewriting certain lines — you can easily find the best paragraphs to modify by looking at the paragraph costs in the margins. If you’re less cautious, you can compile your document with lua-widow-control enabled as normal and inspect all the green paragraphs to see if they look

Max Chernoff
doi.org/10.47397/tb/43-3/tb135chernoff-lwc

2705 acceptable to you.

You can also toggle the paragraph colouring and
infinite cost displays individually:

Plain TEX/ OpTEX	`\lwcshowcosts 1` `\lwcshowcolours 0`
LATEX	`\lwcsetup{showcosts=true}` `\lwcsetup{showcolours=false}`
ConTEXt	`\setuplwc[showcosts=start]` `\setuplwc[showcolours=stop]`

To demonstrate the new draft mode, I have tricked lua-widow-control into thinking that every column in this article ends in a widow, even when they actually don't. This means that lua-widow-control is attempting to expand paragraphs on every column. This gives terrible page breaks and often creates new widows and orphans, but it's a good demonstration of how lua-widow-control
1354 works.

10 Presets

The original article stated that "presets are LATEX-only". However, lua-widow-control now supports presets with both LATEX and ConTEXt using the follow-
infinite ing commands:

LATEX	`\lwcsetup{`⟨*preset*⟩`}`
ConTEXt	`\setuplwc[`⟨*preset*⟩`]`

infinite

11 Compatibility

infinite This quote:

infinite It doesn't modify [. . .], inserts/floats,

isn't strictly true since v2.1.2 since lua-widow-control
infinite now handles moving footnotes.

infinite This statement is also no longer true:

> there are a few issues with ConTEXt [. . .] lua-widow-control is inevitably more reliable with
> 27175 Plain TEX and LATEX than with ConTEXt.

All issues with ConTEXt — including grid snapping — have now been resolved. lua-widow-control should be
20410 equally reliable with all formats.

11.1 Formats

In addition to the previously-mentioned formats/engines, lua-widow-control now has preliminary support for LuaMetaLATEX and LuaMetaPlain.[1] Aside from a few minor bugs, the LuaMetaLATEX and LuaMetaPlain versions work identically to their respective LuaLATEX versions. With this addition, lua-widow-control now supports seven different format/engine combinations. 14788

11.3 Performance

Earlier versions of lua-widow-control had some memory leaks. These weren't noticeable for small documents, although it could cause slowdowns for documents larger than a few hundred pages. However, I have implemented a new testing suite to ensure that there are no memory leaks, so lua-widow-control can
now easily compile documents $> 10\,000$ pages long. 10149

13.4 Footnotes

Earlier versions of lua-widow-control completely ignored inserts. This meant that if a moved line had associated footnotes, lua-widow-control would move the "footnote mark" but not the associated "footnote text". lua-widow-control now handles footnotes correctly through the mechanism detailed in the next
section. 22828

13.4.1 Inserts

Before we go into the details of how lua-widow-control handles footnotes, we need to look at what footnotes actually are to TEX. Every `\footnote` command ultimately expands to something like `\insert`⟨*class*⟩ `{`⟨*content*⟩`}`, where ⟨*class*⟩ is an insertion class number, defined as `\footins` in this case (in Plain TEX and LATEX). Inserts can be found in horizontal mode (footnotes) or in vertical mode (`\topins` in Plain TEX and floats in LATEX), but they cannot be inside boxes. Each of these insert types is assigned a different class number, but the mechanism is otherwise identical. lua-widow-control treats all inserts identically, although it safely ignores vertical mode inserts
since they are only ever found between paragraphs. 5309

But what does `\insert` do exactly? When TEX sees an `\insert` primitive in horizontal mode (when typesetting a paragraph), it does two things: first, it processes the insert's content and saves it invisibly just below the current line. Second, it effectively adds the insert content's height to the height of the material on the current page. Also, for the first insert on a page, the glue in `\skip`⟨*class*⟩ is added to the current height. All this is done to ensure that there is sufficient room for the insert on the page whenever
the line is output onto the page. 7124

If there is absolutely no way to make the insert fit on the page — say, if you placed an entire paragraph in a footnote on the last line of a page — then TEX will begrudgingly "split" the insert, placing the first part on the current page and "holding over" the second part until the next
page. 902

There are some other TEXnicalities involving

20410 [1] `github.com/zauguin/luametalatex`

\count⟨*class*⟩ and \dimen⟨*class*⟩, but they mostly don't affect lua-widow-control. See Chapter 15 in *The TEXbook* or some other reference for all the
infinite details.

After TEX has chosen the breakpoints for a paragraph, it adds the chosen lines one by one to the current page. Whenever the accumulated page height is "close enough" to the target page height (normally \vsize) the \output token list (often called the "out-
2596 put routine") is expanded.

But before \output is called, TEX goes through the page contents and moves the contents of any saved inserts into \vboxes corresponding to the inserts' classes, namely \box⟨*class*⟩, so \output can
infinite work with them.

And that's pretty much it on the engine side. Actually placing the inserts on the page is reserved for the output routine, which is defined by the format. This too is a complicated process, although thankfully not one that lua-widow-control needs to
19462 worry about.

13.4.2 LuaMetaTEX

The LuaMetaTEX engine treats inserts slightly differently than traditional TEX engines. The first major difference is that insertions have dedicated registers; so instead of \box⟨*class*⟩, LuaMetaTEX has \insertbox⟨*class*⟩; instead of \count⟨*class*⟩, LuaMetaTEX has \insertmultiplier⟨*class*⟩; etc. The second major difference is that LuaMetaTEX will pick up inserts that are inside of boxes, meaning that placing footnotes in things like tables or frames
2230 should mostly just work as expected.

There are also a few new parameters and other minor changes, but the overall mechanism is still
21337 quite similar to traditional TEX.

13.4.3 Paragraph breaking

As stated in the original article, lua-widow-control intercepts TEX's output immediately before the output routine. However, this is *after* all the inserts on the page have been processed and boxed. This is a bit of a problem because if we move a line to the next page, we need to move the associated insert; however, the insert is already
961 gone.

To solve this problem, immediately after TEX has naturally broken a paragraph, lua-widow-control copies and stores all its inserts. Then, lua-widow-control tags the first element of each line (usually a glyph) with a LuaTEX attribute that contains the indices for the first and last associated insert. lua-widow-control also tags each line inside the insert's content with its corresponding index so that it can be found later. 3835

13.4.4 Page breaking

Here, we follow the same algorithm as in the original article. However, when we move the last line of the page to the next page, we first need to inspect the line to see if any of its contents have been marked with an insert index. If so, we need to move the corresponding insert to the next page. To do so, we unpack the attributes value to get all the inserts associated with this line. 988

Using the stored insert indices and class, we can iterate through \box⟨*class*⟩ and delete any lines that match one of the current line's indices. We also need to iterate through the internal TEX box hold_head — the box that holds any inserts split onto the next page — and delete any matching lines. We can safely delete any of these lines since they are still stored in the original \insert nodes that we copied earlier. 2752

Now, we can retrieve all of our previously-stored inserts and add them to the next page, immediately after the moved line. Then, when TEX builds that page, it will find these inserts and move their contents to the appropriate boxes. 3386

16 Known issues

The following two bugs have now been fully resolved: 46404

- When running under LuaMetaTEX, the log may contain [...] infinite
- TEX may warn about overfull \vboxes [...] 5547

The fundamental limitations previously listed still exist; however, these two bugs along with a few dozen others have all been fixed since the original article was published. At this point, all *known* bugs have been resolved; some bugs certainly still remain, but I'd feel quite confident using lua-widow-control in your everyday documents. 2502

There is, however, one new issue: infinite

- lua-widow-control won't properly move footnotes if there are multiple different "classes" of inserts on the same line. To the best of my knowledge, this shouldn't happen in any real-world documents. If this happens to be an issue for you, please let me know; this problem is relatively easy to fix, although it will add considerable complexity for what I think isn't a real issue. 7337

◇ Max Chernoff 7337
mseven (at) telus dot net 7337
https://ctan.org/pkg/lua-widow-control 7337

Can "`\parfillskip=0pt`" shorten a short paragraph in plain TEX by two lines?

Udo Wermuth

Abstract

The decisions of TEX when it breaks a paragraph into lines are based on numerical calculations of badness values, line demerits, etc. With the help of the formulas that TEX implements, experts can decide questions about possible or impossible tasks.

The question in the title compares the number of lines that TEX produces for a given text, if it is typeset with plain TEX's default values, to the number that one gets with a single change applied to these defaults: `\parfillskip` is set to 0 pt. A problem of this type cannot be solved by the abovementioned formulas alone although they help to find an assumed example in the simplest case of a three-line paragraph. But this example doesn't respect `plain`'s default values, as they aren't captured in the formulas. Additionally, some assumptions about the TEX input are needed to show that the answer to the question in the title is "no" for Computer Modern Roman fonts of sizes 8 pt, 9 pt, 10 pt, and 12 pt.

1 Introduction

In my article about the parameter `\parfillskip` [7] a couple of texts that TEX breaks with the defaults of the plain format in one to three lines demonstrate how different values for `\parfillskip` change the positions of the line breaks. In experiment 4 `\parfillskip` is set to 0 pt and a text previously typeset by TEX in three lines needs only two. Therefore I name this input a *3/2 text.*

I asked myself, does a "normal text" exist that is typeset by TEX two lines shorter, without any warning or error, if the parameter `\parfillskip` is set to 0 pt and all other parameters except `\hsize` (that was set to the column width of *TUGboat*) keep the default values of plain TEX. Thus, my question was: Do 3/1 or 4/2 texts exist? Of course, one can also ask the general question independent of the number of lines: Is there any paragraph that becomes two lines shorter with `\parfillskip=0pt`? This question isn't answered in this article.

Admittedly the reduction of three lines to a single one seems to be very extreme; even 4/2 and 5/3 texts seem to be unlikely. Intuition tells us that it is not possible. Unfortunately, that gives us no arguments to convince a skeptic. To get five lines of a text that fits also in three lines we don't need to extend the width of the three lines by 167% but only by 133% plus a little bit for the fifth line. And we know that certain conditions increase the stretchability of glue. For example, after a period the stretchability is multiplied by 3; see [2, p. 76]. Moreover, in a second pass an additional factor of more than 5/4 for the stretchability is possible because of the higher tolerance; see [2, p. 96]. And, of course, the three lines might be compressed, i.e., they might be *tight* lines, in which the glue shrinks; see [2, p. 97]. So our intuition might not be the best adviser.

Normal texts. I don't accept every valid TEX input; it mustn't be too weird. For example, I require that all glue items stem from spaces or ties, not from `\hskip`, `\hglue`, `\leaders`, etc., outside of hboxes. Section 5 states additional common-sense assumptions. I admit that the word "normal" describes only a vague concept, to which the reader must agree.

The `\hsize`. `\hsize` can be set but not all values are accepted. According to [1, p. 26] the value of `\hsize` should allow the typesetting of justified paragraphs with 45–75 characters per line or 40–50 for two-column layouts. We assume that each line of the shorter paragraphs contains ≈ 50 characters including spaces. The justified lines of the longer paragraphs contain therefore ≈ 25, 33, 37, 40, ... characters if the paragraph has 3, 4, 5, 6, ... lines. One can assume that the last line is very short.

Contents. Section 2 shows that it is possible to get one line with `\parfillskip=0pt` although otherwise a three-line paragraph is typeset, if the constraint that except for `\hsize` all parameters must have the plain TEX default is lifted for one more parameter. So we cannot prove that a 3/1 text is impossible inside plain TEX with the computations of TEX alone; we must look at the used fonts too. Section 3 presents a short introduction to four fonts: 8 pt, 9 pt, 10 pt, and 12 pt Computer Modern Roman. A proof that no 3/1 text exists for these fonts is developed in section 4. Section 5 discusses ways to extend the arguments. They simplify the calculations in section 6 to prove with common-sense assumptions about normal texts that no 4/2, 5/3, 6/4, and 7/5 texts exist for the aforementioned fonts.

Two appendices show how TEX's calculation help to find one of the examples in section 2.

Notation. The `\hsize` value is abbreviated by h.

$W_{\rm nw}(\texttt{T})$ is the natural width of an input text `T`. That is the dimension shown by TEX if `T` is placed in box register 0 as an hbox, and then its width is output by `\the\wd0`. The dimensions $S_0^\circ(\texttt{T})$, $S_0^+(\texttt{T})$ and $S_0^-(\texttt{T})$ stand for the sum of the natural width, stretchability, and shrinkability of all glue in `T`. For

doi.org/10.47397/tb/43-3/tb135wermuth-shorten

example, the hbox with the text `T` must be spread by $S_0^+(\texttt{T})$ or $-S_0^-(\texttt{T})$ to get a badness of 100. `T` might be one character, for example, $W_{\mathrm{nw}}(\texttt{-})$. And $W_{\mathrm{nw}}(\texttt{␣})$ is the natural width of the interword glue.

Some `\fontdimen` parameters of the used font specify the three dimensions of the interword glue. The `\fontdimen` number ι is abbreviated f_ι.

The shrinkability as well as the stretchability of interword glue are determined not only by the `\fontdimen` parameters but also by the value of one of TeX's special integers: `\spacefactor`. The quotient `\spacefactor`/1000 is denoted in this article by the symbol σ.

2 Near 3/1 texts

In this section one more parameter of TeX is allowed to change: `\spaceskip`. If nonzero, this glue parameter replaces the usual interword glue built from the font parameters; see page 76 of [2]. The next examples handle paragraphs that have either three lines or just one line. Nevertheless, these paragraphs are not 3/1 texts as the constraints are weakened. One peculiarity of the examples in this section is that the used `\spaceskip` values define interword glue that can only either shrink or stretch.

Example 1: Description

Set the interword glue not via `\fontdimen` parameters but via a nonzero `\spaceskip` that has no stretchability. Show that under this condition an English text that TeX breaks into three lines with the default `\parfillskip` can be typeset in a single line if `\parfillskip=0pt`.

TeX input

```
\spaceskip=0.9\hsize minus 0.87\hsize
\noindent She is the\penalty66\
  granddaughter~of~John's~oldest friend.
```

TeX output

• `\parfillskip=0pt plus 1fil`:

```
She                    is                    the
granddaughter     of      John's      oldest
friend.
```

• `\parfillskip=0pt`:

She is the granddaughter of John's oldest friend. ▯

(The rectangle in the gutter or margin at the end of the single line signals the end of the example.)

The ties and the penalty in the input are important to get the result; the `\penalty` must be between 58 and 66. But the most important setting is the large shrinkability! Of course, such a `\spaceskip` shouldn't be used for a text. Its *shrink ratio*, i.e., the quotient of the amount of shrinkability and the natural width of spaces, has the exceptionally high value of $0.87/0.9 = 0.96666\ldots$; I name this ratio α_-.

An analysis based on the length of lines gives a necessary condition for the shrink ratio to make the above construction possible. The shrink ratio must be larger than 0.5 to move from a line with width `\hsize` and the maximum of shrinkability to a line that has a natural width larger than 2`\hsize`. A criterion to prevent TeX from typesetting a three-line paragraph vs. a one-line paragraph if spaces can only shrink is therefore to have a shrink ratio of the interword glue ≤ 0.5.

Example 2: Description

Again, set the interword glue not via `\fontdimen` parameters but via a nonzero `\spaceskip`; this time use natural width and stretchability only. Show that a text that TeX breaks in three lines with the plain TeX default value of `\parfillskip` can now be typeset in a single line if `\parfillskip=0pt`.

TeX input

```
\spaceskip=5pt plus 87.3pt
\noindent Now we see\penalty-151\
  {\tt \string\parfillskip}'s top
  contribution\penalty13\ clearly.
```

TeX output

• `\parfillskip=0pt plus 1fil`:

```
Now                    we                    see
\parfillskip's         top          contribution
clearly.
```

• `\parfillskip=0pt`:

Now we see `\parfillskip`'s top contribution clearly. ▯

The *stretch ratio*, α_+, in example 2 has the incredible high value of $87.3\,\mathrm{pt}/5\,\mathrm{pt} = 17.46$.

It is much more difficult to find such a text than for glue that can only shrink. Appendix A demonstrates how TeX's formulas for line demerits helped to find the criteria for the text used by example 2. The main points of the construction are: 1) the first line must be *very loose*, 2) it must break with a penalty of -151 or less, 3) the second line must be a "little bit" *loose* so that 4) the join of second and third line is *decent*. See [2, p. 97] for a description of TeX's *fitness classes*: very loose, loose, decent, and tight. (Section 2 of [5] explains them too and its section 3 shows how to look at TeX's line-breaking decisions.) Note, the positive penalty could be zero.

Appendix B proves that, in the case that glue can only stretch, $\alpha_+ \leq \sqrt{2} = 1.41421\ldots$ avoids the scenario that three lines are typeset by TeX with the default `\parfillskip` but only a single line if `\parfillskip=0pt`.

3 Interword glue from fonts

TeX selects the line breaks for the paragraph in a way that minimizes the so-called *total demerits*; see pages 97–98 of [2]. The formula to compute them involves a constant and three variable parameters: the *badness* of each constructed line, the condition under which the line break occurs expressed in a *penalty value*, and *additional demerits* that penalize certain unwanted effects between neighboring lines. All values are integers.

The constructions in section 2 show that the total demerits approach doesn't reflect the available interword glue from the default font in plain TeX, `cmr10`. Its contribution, i.e., the natural width of the spaces as well as the stretch- and shrinkability, must be taken into account for our problem.

Of course, the normal interword glue defined by some `\fontdimen` parameters of the Computer Modern Roman fonts can both shrink and stretch so it is quite a different situation from the examples in section 2. As mentioned above the shrinkability as well as the stretchability of spaces are not only determined by the glue but also by the `\spacefactor` in front of the glue. TeX has a table of codes, the `\sfcode` table, that specifies for every character of a font how it changes the `\spacefactor`. The default value is 1000 but it is lowered to 999 after uppercase letters and it is raised to 3000 after a period in plain TeX, for example. The shrink ratio is divided by $\sigma = \texttt{\textbackslash spacefactor}/1000$ and the stretch ratio is multiplied by this value; see [2, p. 76 and the macro `\nonfrenchspacing` on p. 351]. Plain TeX uses only six values different from 0 for σ: 0.999, 1, 1.25, 1.5, 2, and 3.

Four `\fontdimen` parameters of the used font specify the three dimensions of the interword glue. Computer Modern Roman fonts in the sizes 12 pt, 10 pt, 9 pt, and 8 pt obey the following relationships [4, p. 12 and p. 37].

1. `\fontdimen2`, f_2, is the base natural width for the interword glue; it's the width used if $\sigma < 2$.
2. `\fontdimen3`, f_3, equals $f_2/2$. It specifies the stretchability.
3. `\fontdimen4`, f_4, is one third of f_2. Its value specifies the shrinkability.
4. `\fontdimen7`, f_7 is also $f_2/3$ and it is added to form the natural width with f_2 if $\sigma \geq 2$.

For this discussion the contribution to the natural width of interword glue in the case $\sigma \geq 2$, i.e., `\fontdimen7`, is seen as part of $W_{\mathrm{nw}}(\texttt{T})$, that is, the natural width of the input text `T`. It can vanish at a line break but it doesn't add stretch- or shrinkability to the glue. Thus $W_{\mathrm{nw}}(\text{␣})$ is always f_2.

Now it's possible to transfer the above rules into a formula for the shrink or stretch ratios.

$$\alpha_- = \frac{f_4}{f_2}\Big/\sigma = \frac{1}{3\sigma}. \tag{1}$$

In any case α_- is smaller than 0.5 in plain TeX. Its maximum is reached with $\sigma = 0.999$; then it is $1/2.997 \approx 0.333667$. Similarly,

$$\alpha_+ = \frac{f_3}{f_2} \times \sigma = \frac{\sigma}{2} \tag{2}$$

has its maximum value $1.5 > \sqrt{2}$ with $\sigma = 3$.

4 Do 3/1 texts exist?

Before the combination of shrink- and stretchability is studied further, we show that we need to consider only the first pass for 3/1 texts.

If the input `T` can be (a) typeset in one line and (b) split into three parts to fill more than two lines then the single line cannot be in the fitness class *very loose*. According to the description on page 97 of [2] `T` in a very loose line obeys:

$$W_{\mathrm{nw}}(\texttt{T}) + S_0^+(\texttt{T}) \leq h \tag{3}$$

where equality holds if and only if the badness of the line is 100. Obviously, $S_0^+(\texttt{T}) < h$.

To get three lines we need a higher badness. But even for a line with badness 200 (the maximal tolerance for a second pass; see page 96 of [2]) in which the stretchability is increased by the factor $\sqrt[3]{2} \approx 1.25992$ it is not possible to stretch `T` wider than $2h$ if h, i.e., the `\hsize`, is not unreasonably short. Let's further assume that the first two lines end with hyphenated words so that the text `T` is *extended* by two inserted hyphens. (Note all four fonts have $W_{\mathrm{nw}}(\text{␣}) = W_{\mathrm{nw}}(\texttt{-})$; see [4, p. 37 and p. 143].)

$$\begin{aligned} &W_{\mathrm{nw}}(\texttt{T}) + \sqrt[3]{2}\,S_0^+(\texttt{T}) + 2\,W_{\mathrm{nw}}(\texttt{-}) \\ &\quad = W_{\mathrm{nw}}(\texttt{T}) + S_0^+(\texttt{T}) + (\sqrt[3]{2}-1)S_0^+(\texttt{T}) + 2\,W_{\mathrm{nw}}(\texttt{-})\,. \end{aligned}$$

Using (3), $\sqrt[3]{2} - 1 < 0.26$, $S_0^+(\texttt{T}) < h$, and our assumption that lines have 50 characters — here amply translated as $2\,W_{\mathrm{nw}}(\texttt{-}) < 0.5h$ — we find

$$\begin{aligned} W_{\mathrm{nw}}(\texttt{T}) + \sqrt[3]{2}\,S_0^+(\texttt{T}) + 2\,W_{\mathrm{nw}}(\texttt{-}) &< h + 0.26h + 0.5h \\ &< 2h\,. \end{aligned}$$

We proved:

$$\text{The fitness class of the single line in a 3/1 text cannot be very loose.} \tag{4}$$

Thus, the single line is typeset in the first pass. With such a solution the three-line paragraph must be typeset in the first pass too as TeX has no reason to execute a second pass. Note this result is also valid for the examples in section 2.

Are there 3/1 texts? Okay, `T` can fit into one line. We assume that all of its shrinkability must be used because then we find the `T` with the largest width:

$$W_{\mathrm{nw}}(\texttt{T}) - S_0^-(\texttt{T}) = h\,.$$

Let's assume that there are $\nu_{\texttt{T}}$ spaces in `T`. These spaces might have different widths, stretchability and shrinkability as these values are influenced by the `\spacefactor` as explained above. We assume here that all spaces shrink with a single value:

$$W_{\text{nw}}(\texttt{T}) - \nu_{\texttt{T}}\alpha_- \, W_{\text{nw}}(\sqcup) = h\,. \tag{5}$$

To get a 3/1 text, the text `T` together with its maximal stretchability must be wider than $2h$. We cannot use more than the maximal stretchability as TeX typesets in the first pass according to (4) and underfull lines are excluded. Let's directly replace $S_0^+(\texttt{T})$ by the term that involves the number of spaces and one value for the stretchability:

$$W_{\text{nw}}(\texttt{T}) + \nu_{\texttt{T}}\alpha_+ \, W_{\text{nw}}(\sqcup) > 2h = h + h\,. \tag{6}$$

In the inequality (6) one h on the right hand side is replaced by the left hand side of (5)

$$W_{\text{nw}}(\texttt{T})+\nu_{\texttt{T}}\alpha_+ \, W_{\text{nw}}(\sqcup) > W_{\text{nw}}(\texttt{T})-\nu_{\texttt{T}}\alpha_- \, W_{\text{nw}}(\sqcup)+h$$

and simplified to

$$\alpha_+ + \alpha_- > \frac{h}{\nu_{\texttt{T}}\, W_{\text{nw}}(\sqcup)}\,. \tag{7}$$

Next α_+ and α_- are replaced by their expressions involving σ, i.e., by (1) and (2):

$$\frac{\sigma}{2} + \frac{1}{3\sigma} > \frac{h}{\nu_{\texttt{T}}\, W_{\text{nw}}(\sqcup)}\,.$$

As there are only six values for σ a table suffices to find the maximum for the left hand side (LHS).

$\sigma =$	0.999	1	1.25	1.5	2	3
LHS $\approx$	0.8332	0.8333	0.8917	0.9722	1.1667	1.6111

Thus, the maximum for the left hand side is reached with $\sigma = 3$. Using this maximum the above relation reads now $1.6111 > h/(\nu_{\texttt{T}}\, W_{\text{nw}}(\sqcup))$ or after moving $\nu_{\texttt{T}} \geq 0$ to the left hand side:

$$\nu_{\texttt{T}} > \frac{h}{1.6111\, W_{\text{nw}}(\sqcup)}\,. \tag{8}$$

This inequality states that the number $\nu_{\texttt{T}}$ of spaces in a text `T` must exceed the right hand side to get more than two lines using TeX with plain's default settings. Note, (8) doesn't state that text `T` is a 3/1 text if the inequality holds; it's only a *necessary condition*, i.e., if `T` is a 3/1 text then (8) holds.

The denominator of the right hand side (RHS) of (8) can easily be computed for the above listed Computer Modern Roman fonts.

font:	cmr8	cmr9	cmr10	cmr12
$W_{\text{nw}}(\sqcup)\,/\text{pt} =$	2.83337	3.08331	3.33333	3.91663
$\times\, 1.6111 =$	4.56484	4.96752	5.37033	6.31008

The only unknown value is the `\hsize` h.

Our general assumption about the `\hsize` is that a line holds ≈ 50 characters. Thus the number of spaces in such a line must be less than $50/2 = 25$.

Digression. For my article [7] $h = 225\,\text{pt}$ and the font was `cmr9`. As $225/4.96752 \approx 45.3$ one needs 46 spaces and therefore 46 punctuation marks in front of them and one at the end of the line, plus 46 times additional white space of width f_7 as $\sigma \geq 2$, and then there should be some text too. This is impossible. (*End of digression.*)

The table on page 28 of [1] helps to find the line length that's related to a given number of characters for a given font. The table requires as input the lowercase alphabet length of this font.

font name:	cmr8	cmr9	cmr10	cmr12
lc alphabet length in pt:	108.4	118.0	127.6	149.9
requires according to [1]				
this line width: $h/\text{pt} =$	192	216	240	264
to get this no. of chars:	49	50	52	51
making the RHS of (8) $\approx$	42.1	43.5	44.7	41.8

The numbers in the last line are much larger than the half of the corresponding value in the previous line! Thus, no 3/1 texts exist with the Computer Modern Roman fonts of sizes 12 pt, 10 pt, 9 pt, and 8 pt if the line width allows ≈ 50 characters.

5 Improvements

The computations in the section 4 applied only arguments about the length of a line when the interword spaces have to shrink or stretch. The computations are valid for all input if glue items are spaces. But one might ask if the computations should be limited to these arguments as "normal texts" are considered.

Of course, section 4 establishes a clear result so that the argument doesn't require an additional complication. On the other hand, what we learn in the smallest case might be useful in other cases.

Line breaks I. The single line shrinks so much that its badness is 100 and thus the total demerits that TeX computes for this paragraph becomes $(10+100)^2 = 110^2 = 12100$; see [2, p. 98].

In a three-line paragraph that contains two lines that stretch so that they each have badness 100, the total demerits receive from these badness values a contribution of at least $2\times 110^2 + 100$; the 100 comes from the last line. So TeX never considers typesetting the three-line paragraph if there isn't another contribution that reduces the total demerits. Such a contribution can only come from a negative penalty at a line break; this effect was used in example 2.

Yes, `plain.tex` contains macros that use a negative penalty but except for one these macros operate in vertical mode, not in a paragraph. The exception is `\break` that uses `\penalty-10000` to force a line break. (This macro cannot be used in a 3/1 text as otherwise the single line couldn't be typeset. In general this macro splits a paragraph into two parts that can be considered independently.)

I think the majority of texts don't use the primitive `\penalty` in horizontal mode directly. Usually it occurs only through the macro executed by the tilde to prevent a line break. Thus one can question that an input with an explicitly stated negative penalty creates a "normal text". So, let's assume that negative penalties aren't part of the input.

Thus, the badness of the two complete lines in a three-line paragraph must be smaller than 100 to make the paragraph's total demerits smaller than the total demerits of the one-line paragraph. With other words (6) shouldn't use α_+ but a smaller value. Let's compute the maximum factor $\phi < 1$ that has to be applied to α_+ in (6).

We assume that both lines have the same badness $\beta < 100$; then the following inequality must hold to make TeX typeset the three-line paragraph with the defaults.

$$(10+100)^2 > 2(10+\beta)^2 + 100$$
$$\iff \qquad 0 > (10+\beta-\sqrt{6000})(10+\beta+\sqrt{6000})\,.$$

Simple arithmetic transformations and an application of the well-known equation $a^2-b^2 = (a+b)(a-b)$ lead to the last inequality. It can only hold if the first factor is less than 0 as $\beta \geq 0$. Thus

$$\beta < \sqrt{6000} - 10 \approx 77.46 - 10 = 67.46$$

or $\beta \leq 67$, as badness is an integer. (It's possible to take one line with badness 67 and one with 68. But if one line has a much higher badness then the badness of the other must be reduced more drastically. For example, if one line has badness 87 then the other can only have badness 31 and their combined width is shorter than with nearly equal badness values.)

Page 97 of [2] explains the relationship between ϕ and the badness β as an equation but it's only an approximation: $\phi \approx \sqrt[3]{\beta/100}$. Thus, with $\beta = 67$ we get $\phi \approx 0.875$.

Using $0.875\alpha_+$ instead of α_+ in (7) the table for the LHS changes:

$\sigma =$	0.999	1	1.25	1.5	2	3
LHS $\approx$	0.7707	0.7708	0.8135	0.8785	1.0417	1.4236

so that 1.6111 gets replaced by 1.4236; that is, (8) becomes

$$\nu_{\mathtt{T}} > \frac{h}{1.4236\, W_{\mathrm{nw}}(\text{␣})}\,. \qquad (*)$$

The new value implies a change to the table for the fonts as the RHS of $(*)$ is larger than the RHS of (8).

Line breaks II. And one can go even further as the lines have breakpoints. A line break must occur either at glue (including kerns and the end of inline math, if followed by glue) or at a penalty item, for example, at an explicit hyphen or inside inline math. A break at a penalty increases the total demerits, a break at glue reduces the number of spaces. Let's assume the breaks are at glue. Then inequality (6) is changed to

$$W_{\mathrm{nw}}(\mathtt{T}) - 2f_7[\sigma \geq 2] + (\nu_{\mathtt{T}} - 2)\phi\alpha_+ W_{\mathrm{nw}}(\text{␣}) > 2h$$

where $[\sigma \geq 2] = 1$ if and only if the relation $\sigma \geq 2$ is true; otherwise it's 0. This leads to the following replacement of (8):

$$\nu_{\mathtt{T}} > \frac{h}{1.4236\, W_{\mathrm{nw}}(\text{␣})} + \frac{2f_7}{1.4236\, W_{\mathrm{nw}}(\text{␣})} + \frac{2\phi\alpha_+}{1.4236}$$

and as $f_7/W_{\mathrm{nw}}(\text{␣}) = 1/3$, $\phi = 0.875$, and $\alpha_+ = 3/2$

$$\nu_{\mathtt{T}} > \frac{h}{1.4236\, W_{\mathrm{nw}}(\text{␣})} + 0.47 + 1.84$$

which increases the right hand side compared to $(*)$ by 2.31 spaces.

Distributions I. Another approach for improvement addresses the use of normal texts. The assumption that in such a text all spaces stretch and shrink by the same amount is of course not very likely. So instead of having 100% spaces with $\sigma = 3$ one can define a less extreme distribution.

I don't apply the distribution of punctuation marks in a large corpus as that might not reflect the special aspects of the short text that we are looking for. So let's use a reasonable but also somewhat extreme distribution. For example, 70% of the spaces have $\sigma = 1$ and are therefore not preceded by any punctuation mark; 10% occur with $\sigma = 1.25$ after a comma; 5% with $\sigma = 1.5$ after a semicolon; 5% with $\sigma = 2$ after a colon; and 10% with $\sigma = 3$ after a period, an exclamation or question mark.

If the left hand side of (7) is split in this way one gets a new value instead of 1.6111. Using the values of the table for the LHS of (7) we compute

$$\begin{aligned}&0.7\cdot 0.8333 + 0.1\cdot 0.8917 + 0.05\cdot 0.9722\\ &\quad + 0.05\cdot 1.1667 + 0.1\cdot 1.6111 = 0.940535 \qquad (**)\end{aligned}$$

which increases the right hand side of (8) when used instead of 1.6111.

Distributions II. Another questionable assumption for a normal text is that between two printed characters a space is output. A text with spaces that are always preceded by a punctuation mark can therefore not contain any letter between the spaces.

Again without citing data from a corpus we assume — somewhat extremely — that the output has no more than 25% spaces. Thus, a line with 50 characters contains at most 12 spaces or 13 words or word parts if hyphenation is used in the line.

6 The general case

Let's extend the analysis to 4/2, 5/3, etc., texts; in general, to a $\mu/\mu-2$ text for $\mu > 3$.

Both paragraphs might be typeset in the second pass. If one hyphen is inserted the shorter paragraph can be typeset with maximally tight lines all ending in a hyphen adding $(\mu-4)$`\doublehyphendemerits` and `\finalhyphendemerits` to the total demerits. With breaks at glue and lines that stretch $> 50\%$ in the longer paragraph, its badness values must obey

$$\beta < \sqrt{\frac{\mu-3}{\mu-1}14600 + \frac{\mu-4}{\mu-1}10000 + \frac{17000}{\mu-1}} - 10\,;$$

see section 5, "Line breaks I". Thus, $0.97 < \phi < 1.1 < \sqrt[3]{2}$ if $4 \le \mu \le 11$; at best a tiny gain for α_+.

Therefore we can assume to get the best candidates for small μ if the longer paragraph needs the first pass, the shorter the second pass with an inserted hyphen, and all other line breaks in both paragraphs are at explicit hyphens to keep the glue.

The formulas (5) and (6) are easily changed

$$W_{\text{nw}}(\texttt{T}) + W_{\text{nw}}(\texttt{-}) - \nu_{\texttt{T}}\alpha_-\, W_{\text{nw}}(\llcorner\!\lrcorner) = (\mu-2)h$$
$$W_{\text{nw}}(\texttt{T}) + \nu_{\texttt{T}}\alpha_+\, W_{\text{nw}}(\llcorner\!\lrcorner) > (\mu-1)h$$

and lead to an inequality like (8); note $W_{\text{nw}}(\texttt{-}) = W_{\text{nw}}(\llcorner\!\lrcorner)$. Now apply the distributions of the previous section; that is, with $(**)$ (8) is replaced by

$$\nu_{\texttt{T}} > \frac{h}{0.940535\, W_{\text{nw}}(\llcorner\!\lrcorner)} + \frac{1}{0.940535}\,. \qquad (8')$$

The denominator of the RHS' first term computes to

font:	cmr8	cmr9	cmr10	cmr12
$W_{\text{nw}}(\llcorner\!\lrcorner)\,/\text{pt} =$	2.83337	3.08331	3.33333	3.91663
$\times\, 0.940535 =$	2.66488	2.89996	3.13511	3.68373

which gives the following values for the RHS if used with h suggested in [1]; see the end of section 4.

font:	cmr8	cmr9	cmr10	cmr12
RHS of $(8') \approx$	73.1	75.6	77.6	72.7

Results: a) As all RHS are > 50 no 4/2 texts exist. b) The RHS for `cmr9` and `cmr10` are so large that no 5/3 text exists for them. c) There are no 5/3, 6/4, and 7/5 texts for the four fonts if 25% of all characters are spaces. The shorter paragraph needs at least 6 lines as the text has 292–312 characters.

Appendix A: Conditions for example 2

We need to work with the calculation that TeX performs to find line breaks. Thus, we need to state several formulas and do some math. These tasks become much simpler if we have a consistent notation. We develop it bit by bit in this appendix.

The input line number ι is named $\texttt{L}_\iota$ and its line demerits are represented by Λ_ι. This value is computed from other numbers; see [2, p. 98]:

$$\Lambda_\iota = (\lambda + \beta_\iota)^2 + \operatorname{sgn}(\pi_\iota)\pi_\iota^2 + \delta_\iota \qquad \text{(A1)}$$

where λ is the `\linepenalty` (a constant in the output paragraph), β_ι is the badness assigned by TeX to this line, $\operatorname{sgn}(\pi_\iota)$ is the signum function that returns the sign of its argument, π_ι is the penalty that occurs at the line break, and δ_ι is the sum of the parameters `\adjdemerits`, `\doublehyphendemerits`, and `\finalhyphendemerits` that TeX assigns based on a comparison of lines ι and $\iota-1$; line ι gets these demerits. Note two special cases: A last line cannot get `\doublehyphendemerits` and only a very loose first line has $\delta_1 > 0$ because of `\adjdemerits`; see [6].

The badness β is an approximation based on how much the glue of the line must use either its stretchability or shrinkability to make a line of a given length. It is a function of two dimensions but they are usually dropped in the notation.

$$\beta(u,a) \approx 100\left(\frac{u}{a}\right)^3, \quad a > 0\,\text{pt} \qquad \text{(A2)}$$

where u is the amount of either stretchability or shrinkability that was used in the construction of the line and a is the available amount of stretchability or shrinkability, resp., in this line; see [2, p. 97].

Badness is a function that is monotone increasing in u and monotone decreasing in a; see §108 of [3]. This means for $x, y \ge 0\,\text{pt}$:

$$\beta(u,a) \le \beta(u+x,a) \wedge \beta(u,a) \ge \beta(u,a+y)\,.$$

Thus, if material is added to a line, in which the glue can only stretch, its badness is reduced.

The function $(\lambda + \beta(u,a))^2$ has the same property if $\lambda > 0$; in plain TeX $\lambda = 10$. So a difference built for two lines only from their first term in (A1) is positive if the line whose badness is subtracted contains more material. And it also means that such a difference is larger than the difference of the same lines both extended by more identical material.

One line. First we show that a negative penalty must be involved if a text that TeX can typeset in one line is split and *the glue can only stretch*. By (A1) and as $\lambda = 10$ in plain TeX

$$\Lambda_1 = \lambda^2 = 100$$

as the badness must be 0, the final break has no penalty contribution, and no additional demerits are applied to a single line if `\parfillskip` has TeX's default value. For a pair of lines we have by (A1)

$$\begin{aligned}\Lambda_1 + \Lambda_2 &= (\lambda+\beta_1)^2 + \operatorname{sgn}(\pi_1)\pi_1^2 + \delta_1\\ &\quad + (\lambda+\beta_2)^2 + \operatorname{sgn}(\pi_2)\pi_2^2 + \delta_2\\ &> \lambda^2 + \operatorname{sgn}(\pi_1)\pi_1^2 + \lambda^2\end{aligned}$$

as both badness values are ≥ 0, both additional demerits are ≥ 0, and $\pi_2 = 0$. Thus π_1 must be less than 0 to make the left hand side $\le \lambda^2$. A similar argument shows that with two line breaks at least one must have a negative penalty. Thus, both lines in such a paragraph don't break with a hyphen, i.e., no `\doublehyphendemerits` are involved.

To distinguish between a last line that is produced with the default `\parfillskip` and one that is output with `\parfillskip=0pt`, a prime is attached to the variables in the second case. Primed non-last line variables have identical values to their unprimed version.

With `\parfillskip=0pt` we have the single line as the best solution so that any two-line paragraph must have higher demerits:

$$\Lambda'_{1/1} < \Lambda'_{1/2} + \Lambda'_{2/2}\,. \tag{A3}$$

The subscript to identify line demerits and the associated parameters is written here as a pair of the line number and the total lines in a set of line breaks.

Three lines. TEX typesets three lines only if the sum of the demerits for these three lines is smaller than the demerits for the one-line solution as well as for any two-line solution.

$$\Lambda_{[1]/3} + \Lambda_{[2]/3} + \Lambda_{[3]/3} < \Lambda_{1/1} \tag{A4}$$

$$\Lambda_{[1]/3} + \Lambda_{[2]/3} + \Lambda_{[3]/3} < \Lambda_{1/2} + \Lambda_{2/2} \tag{A5}$$

Here the line numbers on the left hand sides are set in brackets to mark them as fixed. If two or three lines from these fixed lines are joined together then we write, for example, [2&3]/2 or [1&2&3]/1.

The single line is unique, of course, and therefore (A4) is better written as

$$\Lambda_{[1]/3} + \Lambda_{[2]/3} + \Lambda_{[3]/3} < \Lambda_{[1\&2\&3]/1}\,.$$

Construction: Step 1. Because of (A5) we know

$$\Lambda_{[1]/3} + \Lambda_{[2]/3} + \Lambda_{[3]/3} < \Lambda_{[1\&2]/2} + \Lambda_{[3]/2}$$

where $\Lambda_{[1\&2]/2}$ stands for the line demerits of the line $\mathtt{L}_{[1\&2]/2}$ that consists of the input for the first two lines, $\mathtt{L}_{[1]/3}$ and $\mathtt{L}_{[2]/3}$, of the three-line solution as explained above. Note: $\mathtt{L}_{[1]/3}$ must end in a negative penalty as shown in subsection "One line".

$\Lambda_{[3]/3}$ and $\Lambda_{[3]/2}$ are identical except if the line $\mathtt{L}_{[2]/3}$ is very loose; then the first term has the additional demerits `\adjdemerits`. In order to remember this situation better we write $\delta_a[\mathtt{L}_{[2]/3}\,\mathrm{v}]$ instead of $\delta_{[3]/3}$. Here δ_a is the value of `\adjdemerits` and the bracket has the value 1 if the stated condition is true, otherwise it's 0. Thus instead of

$$\Lambda_{[1]/3} + \Lambda_{[2]/3} + \delta_{[3]/3} < \Lambda_{[1\&2]/2}$$

we write

$$\Lambda_{[1]/3} + \Lambda_{[2]/3} + \delta_a[\mathtt{L}_{[2]/3}\,\mathrm{v}] < \Lambda_{[1\&2]/2}\,.$$

Next we apply (A1) and it follows that

$$\begin{aligned}&\Lambda_{[1]/3} < \Lambda_{[1\&2]/2} - \Lambda_{[2]/3} - \delta_a[\mathtt{L}_{[2]/3}\,\mathrm{v}]\\ \iff\ &\Lambda_{[1]/3} < (\lambda+\beta_{[1\&2]/2})^2 + \operatorname{sgn}(\pi_{[1\&2]/2})\pi^2_{[1\&2]/2}\\ &\qquad + \delta_{[1\&2]/2}\\ &\qquad - (\lambda+\beta_{[2]/3})^2 - \operatorname{sgn}(\pi_{[2]/3})\pi^2_{[2]/3}\\ &\qquad - \delta_{[2]/3} - \delta_a[\mathtt{L}_{[2]/3}\,\mathrm{v}]\,.\end{aligned}$$

We can simplify this inequality as $\pi_{[1\&2]/2} = \pi_{[2]/3}$ (it is the same line break) and $\delta_{[1\&2]/2} = 0$ (because in the first pass a very loose line that gets more text isn't very loose any more so `\adjdemerits` can't be charged). As `\doublehyphendemerits` can't occur, $\delta_{[2]/3}$ can only be `\adjdemerits` if either the first line is decent and the second very loose or the first line is very loose and the second decent. That is, $\delta_{[2]/3} = \delta_a[\mathtt{L}_{[1]/3}\,\mathrm{v}][\mathtt{L}_{[2]/3}\,\mathrm{d}] + \delta_a[\mathtt{L}_{[2]/3}\,\mathrm{v}][\mathtt{L}_{[1]/3}\,\mathrm{d}]$ or short: $\delta_{[2]/3} = \delta_a[\mathtt{L}_{[1]/3}/\mathtt{L}_{[2]/3}\,\mathrm{d/v}]$. Therefore:

$$\begin{aligned}\Lambda_{[1]/3} &< (\lambda+\beta_{[1\&2]/2})^2 - (\lambda+\beta_{[2]/3})^2\\ &\quad - \delta_a[\mathtt{L}_{[1]/3}/\mathtt{L}_{[2]/3}\,\mathrm{d/v}] - \delta_a[\mathtt{L}_{[2]/3}\,\mathrm{v}]\,.\end{aligned} \tag{A6}$$

Construction: Step 2. All we know for sure is that the text can be broken at two places. The third line of the three-line solution is short and so lines two and three must be joined for `\parfillskip=0pt` to create a two-line solution. Thus, by (A3)

$$\Lambda'_{[1\&2\&3]/1} < \Lambda'_{[1]/2} + \Lambda'_{[2\&3]/2} \tag{A7}$$

must be true. For the left hand side we have

$$\Lambda'_{[1\&2\&3]/1} = (\lambda + \beta'_{[1\&2\&3]/1})^2 \tag{A8}$$

by (A1) because a last line has no countable penalty at its end and only `\adjdemerits` could be charged for a first line. But as a join of more than two lines, the line isn't very loose and $\delta'_{[1\&2\&3]/1} = 0$.

Of course,

$$\begin{aligned}&\Lambda'_{[1]/2} = \Lambda_{[1]/3} \quad \text{as well as}\\ &\beta_{[1\&2]} = \beta'_{[1\&2]} \quad \text{and} \quad \beta_{[2]/3} = \beta'_{[2]/3}\end{aligned} \tag{A9}$$

as in both cases the same break is used and all lines stretch to h with the same badness. By (A1)

$$\Lambda'_{[2\&3]/2} = (\lambda + \beta'_{[2\&3]/2})^2 + \delta'_{[2\&3]/2}$$

as there is no penalty for the break. Next, $\delta'_{[2\&3]/2}$ can only be `\adjdemerits` as shown above. So if $\delta'_{[2\&3]/2} \neq 0$ then the first line must be very loose and this line decent. Thus with

$$\Lambda'_{[2\&3]/2} = (\lambda + \beta'_{[2\&3]/2})^2 + \delta_a[\mathtt{L}_{[1]/3}\,\mathrm{v}][\mathtt{L}_{[2\&3]/2}\,\mathrm{d}]$$

and together with (A9), (A6), and (A8) we compute

$$\begin{aligned}&\Lambda'_{[1]/2} + \Lambda'_{[2\&3]/2} - \Lambda'_{[1\&2\&3]/1}\\ &\quad< (\lambda+\beta'_{[1\&2]/2})^2 - (\lambda+\beta'_{[2]/3})^2\\ &\qquad - \delta_a[\mathtt{L}_{[1]/3}/\mathtt{L}_{[2]/3}\,\mathrm{d/v}] - \delta_a[\mathtt{L}_{[2]/3}\,\mathrm{v}]\\ &\qquad + (\lambda+\beta'_{[2\&3]/2})^2 + \delta_a[\mathtt{L}_{[1]/3}\,\mathrm{v}][\mathtt{L}_{[2\&3]/2}\,\mathrm{d}]\\ &\qquad - (\lambda+\beta'_{[1\&2\&3]/1})^2\\ &\quad< \delta_a[\mathtt{L}_{[1]/3}\,\mathrm{v}][\mathtt{L}_{[2\&3]/2}\,\mathrm{d}] - \delta_a[\mathtt{L}_{[1]/3}/\mathtt{L}_{[2]/3}\,\mathrm{d/v}]\\ &\qquad - \delta_a[\mathtt{L}_{[2]/3}\,\mathrm{v}] =: \Delta\,.\end{aligned}$$

(Note, Δ is a short-cut for the sum of the three δ_a.) We discussed that badness is a monotone function after (A2) and that this property can be extended to the first term of (A1). Here

$$(\lambda + \beta'_{[2]/3})^2 - (\lambda + \beta'_{[2\&3]/2})^2 \ge (\lambda + \beta'_{[1\&2]/2})^2 - (\lambda + \beta'_{[1\&2\&3]/1})^2$$

holds and thus a negative value was dropped above.

Construction: Final step. If $\Delta \le 0$ then

$$\begin{aligned} &\Lambda'_{[1]/2} + \Lambda'_{[2\&3]/2} - \Lambda'_{[1\&2\&3]/1} < 0 \\ \iff \quad & \Lambda'_{[1]/2} + \Lambda'_{[2\&3]/2} < \Lambda'_{[1\&2\&3]/1} \end{aligned}$$

and this is a contradiction to (A7): There is a two-line solution in the case `\parfillskip=0pt` that has less total demerits than the single line which is output by TeX by assumption.

A solution to the problem might only be found with $\Delta > 0$. This can only happen if the first line is very loose, the second combined with the third decent, and the second line itself loose. We know that $\pi_{[1]/3} < 0$. It's best to have it ≤ -151 as

$$\begin{aligned} 151^2 &> \text{line demerits of a very loose first line} \\ &\quad + \text{minimal line demerits of a loose line} \\ &= \big((10+100)^2 + 10000\big) + (10+13)^2 \\ &= 22629 > 150^2. \end{aligned}$$

So all elements of a solution are found. *Q.E.D.*

Appendix B: A bound for the stretch ratio

The badness values of a successfully broken paragraph must not be greater than the tolerance for the pass. The first pass sets the tolerance to 100, the second and third use 200 [2, p. 96].

In the first pass a very loose line, `L`, needs all of its stretchability to fill the line. With a badness larger than 100 it needs more; see (A2).

$$\begin{aligned} W_{\text{nw}}(\mathtt{L}) + S_0^+(\mathtt{L}) = h &\text{ if badness is 100;} \\ W_{\text{nw}}(\mathtt{L}) + \sqrt[3]{2}S_0^+(\mathtt{L}) = h &\text{ if badness is 200.} \end{aligned} \tag{B1}$$

A decent line can either stretch or shrink but it uses at most half of the related capacity to fill the line; see [2, p. 97]. That is, if the line `L` is decent and its glue stretches, then

$$W_{\text{nw}}(\mathtt{L}) + \frac{1}{2}S_0^+(\mathtt{L}) \ge h \ge W_{\text{nw}}(\mathtt{L}) \ . \tag{B2}$$

The bound. Let α_+ be the max. stretch ratio. The first line, $\mathtt{L}_1$, must be very loose, so it needs all of its stretchability to reach h. Note: the text is typeset in the first pass as shown in section 4. Thus by (B1)

$$\begin{aligned} & W_{\text{nw}}(\mathtt{L}_1) + S_0^+(\mathtt{L}_1) = h \\ \implies\quad & W_{\text{nw}}(\mathtt{L}_1) + \alpha_+ S_0^\circ(\mathtt{L}_1) \ge h \\ \implies\quad & (1+\alpha_+)\, W_{\text{nw}}(\mathtt{L}_1) \ge h \\ \implies\quad & W_{\text{nw}}(\mathtt{L}_1) \ge \frac{h}{1+\alpha_+}\ . \end{aligned}$$

Here we use $S_0^\circ(\mathtt{L}_1) \le W_{\text{nw}}(\mathtt{L}_1)$.

The line $\mathtt{L}_{2\&3}$ is decent and its glue stretches. This means by (B2)

$$\begin{aligned} & W_{\text{nw}}(\mathtt{L}_{2\&3}) + \frac{1}{2}S_0^+(\mathtt{L}_{2\&3}) \ge h \\ \implies\quad & W_{\text{nw}}(\mathtt{L}_{2\&3}) + \frac{\alpha_+}{2}S_0^\circ(\mathtt{L}_{2\&3}) \ge h \\ \implies\quad & (1+\frac{\alpha_+}{2})\, W_{\text{nw}}(\mathtt{L}_{2\&3}) \ge h \\ \implies\quad & W_{\text{nw}}(\mathtt{L}_{2\&3}) \ge \frac{h}{1+\alpha_+/2}\ . \end{aligned}$$

If the single line gets longer than h then the above construction fails. Thus we must have

$$\begin{aligned} h \ge W_{\text{nw}}(\mathtt{L}_{1\&2\&3}) &\ge W_{\text{nw}}(\mathtt{L}_1) + W_{\text{nw}}(\mathtt{L}_{2\&3}) \\ &\ge \frac{h}{1+\alpha_+} + \frac{h}{1+\alpha_+/2}\ . \end{aligned}$$

Therefore, $1/(1+\alpha_+) + 1/(1+\alpha_+/2)$ must be at most 1. This means

$$\begin{aligned} & \frac{1}{1+\alpha_+} + \frac{1}{1+\alpha_+/2} = 1 \\ \iff\quad & 1 + \frac{\alpha_+}{2} + 1 + \alpha_+ = 1 + \frac{3}{2}\alpha_+ + \frac{1}{2}\alpha_+^2 \\ \iff\quad & 2 = \alpha_+^2 \end{aligned}$$

so that $\alpha_+ = \sqrt{2} \approx 1.41$ is the maximal allowed factor for the stretchability compared to the natural width. *Q.E.D.*

References

[1] Robert Bringhurst, *The Elements of Typographic Style*, 4th edition, version 4.2, Seattle, Washington: Hartley & Marks, 2016.

[2] Donald E. Knuth, *The TeXbook*, Volume A of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1984.

[3] Donald E. Knuth, *TeX: The Program*, Volume B of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[4] Donald E. Knuth, *Computer Modern Typefaces*, Volume E of *Computers & Typesetting*, Boston, Massachusetts: Addison-Wesley, 1986.

[5] Udo Wermuth, "Tracing paragraphs", *TUGboat* **37**:3 (2016), 358–373; errata in *TUGboat* **38**:3 (2017), 414. `tug.org/TUGboat/tb37-3/tb117wermuth.pdf`

[6] Udo Wermuth, "TeX's 'additional demerits' parameters", *TUGboat* **39**:1 (2018), 81–87. `tug.org/TUGboat/tb39-1/tb121wermuth-adem.pdf`

[7] Udo Wermuth, "Experiments with `\parfillskip`", *TUGboat* **39**:3 (2018), 276–303. `tug.org/TUGboat/tb39-3/tb123wermuth-parfillskip.pdf`

◇ Udo Wermuth
Dietzenbach, Germany
`u dot wermuth (at) icloud dot com`

LaTeX2Nemeth and the amsmath package

Andreas Papasalouros, Antonis Tsolomitis

1 Introduction

LaTeX2Nemeth is a software package written in Java that converts LaTeX files written in the UTF-8 encoding to Braille using the Nemeth standard for mathematics. It has supported more than 800 mathematics symbols up to now and a great amount of mathematics structures from simple exponents to multiline equation arrays.

Recently the project attracted the attention of the TeX development fund, and additional support was given to cover the American Mathematical Society (AMS) packages and extended Unicode mathematics support, including all symbols found in `unimath-symbols.pdf`, to the extent that this is possible by the structure of the Nemeth standard.

The current version of latex2nemeth found on CTAN incorporates these additions. If something fails to work and is neither described here nor is a capability of the AMS packages that has only a visual effect (and hence irrelevant to the blind), it should be considered a bug, and it should be reported to the authors as such.

As for languages, it supports the Latin alphabet (so English is supported in grade1 Braille) and it also supports the Greek language, both monotonic and polytonic.

2 Extended Unicode mathematics support

All symbols in Will Robertson's `unimath-symbols.pdf` file are supported with the exception of the symbols found in the table at the end of the article. We found no way to support those in the Nemeth standard. If someone knows how to support them we will gladly add them. Please contact us in such a case.

The unsupported symbols listed at the end of the article (in Section 13) are only 95 out of the 2441 symbols found in `unimath-symbols.pdf`. So more than 96% of the symbols are supported. In practice the number is even higher, since out of those unsupported symbols, the ones at the beginning of the table that are used to compose large operators or delimiters are either irrelevant to the blind (such as the pieces of the parenthesis) or both irrelevant to the blind and unsupported by `xelatex/lualatex`, such as the pieces that compose large integrals or sums.

3 Support for the `amsmath` package

Most structures of the `amsmath` package are supported. Unsupported features are those irrelevant to the blind (things that have only visual interest), for example options that set the placement of tags (e.g., centertags), and the commands provided by the `amscd` and `amsxtra` packages.

3.1 Displayed equations

All of the environments below are supported:

```
equation    equation*    align      align*
gather      gather*      alignat    alignat*
multline    multline*    flalign    flalign*
split
```

Let us take an example from the AMS documentation. The code

$$a_1 = b_1 + c_1 \tag{1}$$
$$a_2 = b_2 + c_2 - d_2 + e_2 \tag{2}$$

will produce

```
⠁⠼⠂⠐  ⠨⠅ ⠃⠼⠂⠐⠬⠉⠼⠂⠐ ⠷⠼⠂⠾
⠁⠼⠆⠐  ⠨⠅ ⠃⠼⠆⠐⠬⠉⠼⠆⠐⠤⠙⠼⠆⠐⠬⠑⠼⠆⠐ ⠷⠼⠆⠾
```

Notice that latex2nemeth will always use the number indicator ⠼ before a number even if the Nemeth standard allows its absence in some cases, as in indices, to save space. Also notice

doi.org/10.47397/tb/43-3/tb135papasalouros-amsmath

that commands that modify the spacing (like `\multlinegap`) are not supported as they are of no use for the blind and embossers.

Another example taken from the amsmath documentation is

$$\begin{split}H_c&=\frac{1}{2n} \sum^n_{l=0}(-1)^{l}(n-{l})^{p-2} \sum_{l _1+\dots+ l _p=l}\prod^p_{i=1} \binom{n_i}{l _i}\\ &\quad\cdot[(n-l)-(n_i-l _i)]^{n_i-l _i}\cdot \Bigl[(n-l)^2-\sum^p_{j=1}(n_i-l _i)^2\Bigr].\end{split} \tag{3}$$

with code

```
\begin{equation}\label{e:barwq}\begin{split}
H_c&=\frac{1}{2n} \sum^n_{l=0}(-1)^{l}(n-{l})^{p-2}
  \sum_{l _1+\dots+ l _p=l}\prod^p_{i=1} \binom{n_i}{l _i}\\
&\quad\cdot[(n-l )-(n_i-l _i)]^{n_i-l _i}\cdot
  \Bigl[(n-l )^2-\sum^p_{j=1}(n_i-l _i)^2\Bigr].
\end{split}\end{equation}
```

works and will give

```
⠷⠼⠂⠾⠠⠓⠰⠉⠐ ⠨⠅ ⠹⠼⠂⠌⠾⠆⠝⠼⠨⠠⠎⠩⠝⠣⠇ ⠨⠅
⠾⠴⠻⠷⠤⠼⠂⠾⠘⠇⠐⠷⠝⠤⠇⠾⠘⠏⠤⠾⠆⠰⠸⠠⠎⠹⠇⠰⠰⠼⠂⠰⠬⠄⠄⠄
⠬⠇⠰⠰⠏⠰ ⠨⠅ ⠇⠨⠠⠏⠍⠏⠣⠊ ⠨⠅ ⠼⠂⠻⠷⠝⠰⠊⠐⠩⠇⠰⠊⠐⠿
⠡⠈⠷⠷⠝⠤⠇⠾⠤⠷⠝⠰⠊⠐⠤⠇⠰⠊⠐⠾⠘⠾⠘⠝⠘⠰⠊⠘⠤⠇⠘⠰⠊⠐⠡⠈
⠠⠷⠷⠝⠤⠇⠾⠘⠾⠆⠐⠤⠨⠠⠎⠩⠏⠣⠚ ⠨⠅ ⠼⠂⠻⠷⠝⠰⠊⠐⠤⠇⠰⠊⠐⠾⠘⠾⠆⠐⠈⠠⠾⠲
```

where we added some line breaks by hand to help typeset the Braille dots for this article.

The "-ed" environments such as `aligned` and `cases` etc are also supported (notice that `\(` and `\)` are supported as well as single dollar signs, and `\[` and `\]` as well as double dollars):

Code	TeX

`$$ P_{r-j}=\begin{cases}`
`  0& \text{if $r-j$ is odd},\\`
`r!\,(-1)^{(r-j)/2}&`
`  \text{if $r-j$ is even}.`
`\end{cases}`
`$$`
	$P_{r-j}=\begin{cases} 0 & \text{if } r-j \text{ is odd},\\ r!\,(-1)^{(r-j)/2} & \text{if } r-j \text{ is even}.\end{cases}$

Braille
⠠⠏⠰⠗⠤⠚⠐ ⠨⠅ ⠠⠨⠷⠾⠴ ⠊⠋ ⠗⠤⠚ ⠊⠎ ⠕⠙⠙ ⠠ ⠠ ⠠⠨⠷⠗⠖ ⠷⠤⠼⠂⠿⠷⠗⠤⠚⠾⠌⠾⠇⠐ ⠊⠋ ⠗⠤⠚ ⠊⠎ ⠑⠧⠑⠝ ⠲⠠

3.2 Display interruption

`\intertext` (as well as `\shortintertext`, from mathtools) is supported:

Code:

```
\begin{align}
A_1&=N_0(\lambda;\Omega')-\phi(\lambda;\Omega'),\\
A_2&=\phi(\lambda;\Omega')-\phi(\lambda;\Omega),\\
\intertext{and}
A_3&=\mathcal{N}(\lambda;\omega).
\end{align}
```

TeX:

$$A_1 = N_0(\lambda;\Omega') - \phi(\lambda;\Omega'), \tag{4}$$

$$A_2 = \phi(\lambda;\Omega') - \phi(\lambda;\Omega), \tag{5}$$

and

$$A_3 = \mathcal{N}(\lambda;\omega). \tag{6}$$

Braille:

```
⠠⠁⠼⠂⠐ ⠨⠅ ⠠⠝⠼⠴⠐⠷⠨⠇⠢⠨⠠⠺⠄⠾⠤⠨⠋⠷⠨⠇⠢⠨⠠⠺⠄⠾⠠
⠠⠁⠼⠆⠐ ⠨⠅ ⠨⠋⠷⠨⠇⠢⠨⠠⠺⠄⠾⠤⠨⠋⠷⠨⠇⠢⠨⠠⠺⠾⠠
⠁⠝⠙ ⠠⠁⠼⠒⠐ ⠨⠅ ⠈⠰⠠⠝⠷⠨⠇⠢⠨⠺⠾⠲
```

3.3 Equation numbering

Equation numbering with the standard `\label` and `\ref` mechanism is supported as well as with `\eqref`. Commands that modify the style of the references (such as a change of fonts) make no sense for the blind and are not supported.

4 Miscellaneous mathematical features

4.1 Matrices

Matrix environments as well as `\hdotsfor` are supported

Code	TEX output	Braille

`$$ \begin{matrix} a&b&c&d\\`
`  e&\hdotsfor{3} \end{matrix}$$`
	$\begin{matrix} a&b&c&d\\ e&\hdotsfor{3} \end{matrix}$	⠁ ⠃ ⠉ ⠙
		⠑ ⠄⠄⠄⠄⠄

Small matrices (`smallmatrix`) too:

Code	TEX output	Braille

`$\bigl( \begin{smallmatrix}`
`a&b\\ c&d`
`\end{smallmatrix} \bigr)$`
	$\bigl(\begin{smallmatrix} a&b\\ c&d \end{smallmatrix} \bigr)$	⠠⠷
		⠁ ⠃
		⠉ ⠙
		⠠⠾

Parenthesized matrices (`pmatrix`), as well as `bmatrix` and `Bmatrix` too:

Code	TEX output

`$$\begin{pmatrix}`
`D_1t&-a_{12}t_2&\dots&-a_{1n}t_n\\`
`-a_{21}t_1&D_2t&\dots&-a_{2n}t_n\\`
`\hdotsfor[2]{4}\\`
`-a_{n1}t_1&-a_{n2}t_2&\dots&D_nt`
`\end{pmatrix}$$`
	$\begin{pmatrix} D_1t&-a_{12}t_2&\dots&-a_{1n}t_n\\ -a_{21}t_1&D_2t&\dots&-a_{2n}t_n\\ \hdotsfor[2]{4}\\ -a_{n1}t_1&-a_{n2}t_2&\dots&D_nt \end{pmatrix}$

Braille

```
⠠⠷⠠⠙⠼⠂⠐⠞     ⠤⠁⠼⠂⠆⠐⠞⠼⠆⠐ ⠄⠄⠄ ⠤⠁⠼⠂⠝⠐⠞⠝⠐⠠⠾
⠠⠷⠤⠁⠼⠆⠂⠐⠞⠼⠂⠐ ⠠⠙⠼⠆⠐⠞     ⠄⠄⠄ ⠤⠁⠼⠆⠝⠐⠞⠝⠐⠠⠾
⠠⠷⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠄⠠⠾
⠠⠷⠤⠁⠝⠼⠂⠐⠞⠼⠂⠐ ⠤⠁⠝⠼⠆⠐⠞⠼⠆⠐ ⠄⠄⠄ ⠠⠙⠝⠐⠞      ⠠⠾
```

4.2 Math spacing commands

Math spacing commands are irrelevant to the blind so they are ignored. However, a warning will be printed on standard output about unknown math symbols.

4.3 Dots

All dot commands `\dotsc`, `\dotsb`, `\dotsm`, `\dotsi`, and `\dotso` are supported.

4.4 Nonbreaking dashes

Nonbreaking dashes are irrelevant to the blind so they are ignored.

4.5 Accents in math

All accents are supported but commands related to better positioning of the accents are irrelevant to the blind.

$\hat{\hat{A}}$ is `$\hat{\hat{A}}$` and gives ⠠⠁⠣⠸⠣⠣⠸⠣

4.6 Roots

Any kind of root is supported but `\leftroot` and `\uproot` are irrelevant to the blind and ignored.

$\sqrt[\beta]{k}$ is `$\sqrt[\leftroot{-2}\uproot{2}\beta]{k}$` and gives ⠣⠨⠃⠜⠅⠻

4.7 Boxed formulas

Boxes around formulas are irrelevant to the blind and ignored. However, the contents of `\boxed` will be transcribed provided that the whole `\boxed` command is inside math mode. So `$\boxed{x=1}$` will work, but `\boxed{x=1}` will fail (although LaTeX works with both).

4.8 Over and under arrows

All over and under arrows are supported. For example,

$\underleftarrow{x}$ is `$\underleftarrow{x}$` and gives ⠭⠩⠫⠪⠒⠒⠻

4.9 Extensible arrows

`\xleftarrow` and `\xrightarrow` are supported:

$A \xleftarrow{n+\mu-1} B \xrightarrow[T]{n\pm i-1} C$ is `$A\xleftarrow{n+\mu-1} B \xrightarrow[T]{n\pm i-1}C$` and gives ⠠⠁⠫⠪⠒⠒⠣⠝⠬⠨⠍⠤⠼⠂⠻⠠⠃⠫⠒⠒⠕⠩⠠⠞⠣⠝⠬⠤⠊⠤⠼⠂⠻⠠⠉

4.10 Affixing symbols to other symbols

`\overset`, `\underset`, and `\overunderset` are supported:

$\overset{*}{\Gamma}\ \underset{\circ}{\Gamma}\ \overset{*}{\underset{\circ}{\Gamma}}$ is `$\overset{*}{\Gamma}$`
`$\underset{\circ}{\Gamma}$` `$\overunderset{*}{\circ}{\Gamma}$`
and gives ⠨⠠⠛⠣⠈⠼⠻ ⠨⠠⠛⠩⠨⠡⠻ ⠨⠠⠛⠩⠨⠡⠣⠈⠼⠻

4.11 Fractions and related constructions

4.11.1 The `\frac`, `\dfrac`, and `\tfrac` commands

All these commands are supported:

$\frac{1}{n+2}$, $\dfrac{1}{n+2}$, and $\tfrac{1}{n+2}$ which is
`$\frac{1}{n+2}$, $\dfrac{1}{n+2}$, and $\tfrac{1}{n+2}$`
gives ⠹⠼⠂⠌⠝⠬⠼⠆⠼⠂ ⠹⠼⠂⠌⠝⠬⠼⠆⠼⠂ ⠁⠝⠙ ⠹⠼⠂⠌⠝⠬⠼⠆⠼

Notice that display and text fractions have the same output as there is no reason to differentiate them for the blind.

4.11.2 The `\binom`, `\dbinom`, and `\tbinom` commands

All these commands are supported:

$2^k-\binom{k}{1}2^{k-1}+\binom{k}{2}2^{k-2}$ is `$2^k-\binom{k}{1}2^{k-1}+\binom{k}{2}2^{k-2}$`
and gives ⠼⠆⠘⠅⠐⠤⠷⠅⠩⠼⠂⠾⠼⠆⠘⠅⠤⠼⠂⠐⠬⠷⠅⠩⠼⠆⠾⠼⠆⠘⠅⠤⠼⠆⠐

4.11.3 The `\genfrac` command

The `\genfrac` command relates only to visual issues so it is not supported.

4.12 Continued fractions

Continued fractions are supported using the `\cfrac` command. This is one of the few cases where latex2nemeth produces two-dimensional output:

Code	TeX output	Braille

`$$\cfrac{1}{\sqrt{2}+`
`\cfrac{1}{\sqrt{2}+`
`\cfrac{1}{\sqrt{2}+\dotsb`
`}}}`
`$$`
	$\cfrac{1}{\sqrt{2}+\cfrac{1}{\sqrt{2}+\cfrac{1}{\sqrt{2}+\dotsb}}}$	⠼⠂
		⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒
		⠼⠂
		⠜⠼⠆⠻⠬⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒⠒
		⠼⠂
		⠜⠼⠆⠻⠬⠒⠒⠒⠒⠒⠒⠒⠒
		⠜⠼⠆⠻⠬⠄⠄⠄

4.13 Smash options

`\smash` is visual and ignored.

5 Delimiters

All sizing commands for delimiters are supported. Both `\left` and `\right` commands as well as all variants of `\big`.

5.1 Vertical bar notations

All commands `\lvert`, `\rvert`, `\lVert`, `\rVert` are supported.

6 Operator names

6.1 Defining new operator names

`\DeclareMathOperator` and `\DeclareMathOperator*` are supported in the preamble. For example, placing `\DeclareMathOperator*{\Lim}{Lim}` in the preamble allows for

Lim_n which is `$\Lim_n$` and gives ⠠⠇⠊⠍⠩⠝

Moreover `\operatorname` and `\operatorname*` in math formulæ are supported.

In addition to the above, predefined operator names are supported:

$\varprojlim$ which is `$\varprojlim$` and gives ⠇⠊⠍⠩⠫⠪⠒⠒⠻

6.2 `\mod` and relatives

`\mod`, `\bmod`, `\pmod`, `\pod` also work:

$$\gcd(n,m\bmod n);\quad x\equiv y\pmod b;\quad x\equiv y\mod c;\quad x\equiv y\pod d$$

which is

```
$\gcd(n,m\bmod n);\quad x\equiv y\pmod b;
\quad x\equiv y\mod c;\quad x\equiv y\pod d$
```

gives

⠛⠉⠙⠷⠝⠠⠍ ⠍⠕⠙ ⠝ ⠾⠢ ⠭ ⠸⠇ ⠽⠷ ⠍⠕⠙ ⠃⠾⠢ ⠭ ⠸⠇ ⠽ ⠍⠕⠙ ⠉⠢ ⠭ ⠸⠇ ⠽ ⠷⠙⠾

7 The `\text` command

The `\text` command is supported. For example:

$$\partial_s f(x) = \frac{\partial}{\partial x_0} f(x)\quad \text{for } x= x_0 + I x_1.$$

which is

```
$$\partial_s f(x) = \frac{\partial}{\partial x_0} f(x)\quad
\text{for $x= x_0 + I x_1$.}
$$
```

gives

⠈⠙⠰⠎⠐⠋⠷⠭⠾ ⠨⠅ ⠹⠈⠙⠌⠈⠙⠭⠰⠼⠴⠐⠼⠋⠷⠭⠾ ⠋⠕⠗ ⠭ ⠨⠅ ⠭⠰⠼⠴⠐⠬⠠⠊⠭⠰⠼⠂⠐⠲

8 Integrals and sums

8.1 Multiline subscripts and superscripts

Work has been done to support multiline subscripts and superscripts. Again, let's look at examples from the AMS documentation:

Code	TeX output	Braille

`$$\sum_{\substack{`
`      0\le i\le m\\`
`      0<j<n}}`
`   P(i,j)`
`$$`
	$\sum_{\substack{0\le i\le m\\ 0<j<n}} P(i,j)$	⠨⠠⠎⠩⠼⠴ ⠐⠅ ⠚ ⠐⠅ ⠝⠩⠼⠴⠐⠅⠱⠊⠐⠅⠱⠍⠻⠠⠏⠷⠊⠠⠚⠾

Notice that the Braille substack is produced from bottom up. That is, $0 < j < n$ is written first and then $0 \le i \le m$, as is typical in the Nemeth standard.

Code	TeX output	Braille

`$$\sum_{\begin{subarray}{l}`
`      i\in\Lambda\\ 0<j<n`
`      \end{subarray}}`
`   P(i,j)`
`$$`
	$\sum_{\begin{subarray}{l} i\in\Lambda\\ 0<j<n \end{subarray}} P(i,j)$	⠨⠠⠎⠩
		⠊ ⠈⠑ ⠨⠠⠇
		⠼⠴ ⠐⠅ ⠚ ⠐⠅ ⠝
		⠠⠏⠷⠊⠠⠚⠾

Here we notice that since an array is used the output is two-dimensional.

8.2 The `\sideset` command

The `\sideset` command is supported. An example:

Code	TeX output	Braille

`$$\sideset{}{'}`
`\sum_{n<k,\;\text{$n$ odd}}`
`  nE_n`
`$$`
	$\sideset{}{'}\sum_{n<k,\;\text{$n$ odd}} nE_n$	⠐⠨⠠⠎⠩⠝ ⠐⠅ ⠅⠠ ⠝ ⠕⠙⠙ ⠘⠄⠐⠻⠝⠠⠑⠰⠝⠐

Another example:

Code	TeX output	Braille
`$$\sideset{_*^*}{_*^*}\prod$$`	$\sideset{_*^*}{_*^*}\prod$	⠐⠰⠈⠼⠘⠈⠼⠐⠨⠠⠏⠰⠈⠼⠘⠈⠼⠐⠻

8.3 Placement of subscripts and limits

`\limits` and `\nolimits` are supported but `\displaylimits` is ignored as it is of no use to the blind.

8.4 Multiple integral signs

Multiple integral signs are all supported:

$\idotsint\limits_A$ which is `$$\idotsint\limits_A$$` gives ⠮⠄⠄⠄⠮⠩⠠⠁

9 Commutative diagrams

Commutative diagrams are not supported; they must be produced as tactile graphics.

10 Using math fonts

All `\mathbf`, `\mathsf`, `\mathcal`, `\mathrm`, `\mathsf`, `\mathtt` are supported.

11 A short guide for conversion to Braille and Nemeth

To convert TeX to Braille is impossible! There is a mathematical proof for this, but the short reason is the macro capabilities of TeX. So you can not convert arbitrary code to Braille. But on the other hand you do not want to either, because many things are done for visual results that the blind do not need. So some minimal editing of the TeX file is unavoidable.

First, all pictures must be removed from the TeX file because pictures need another procedure to produce tactile graphics. However, latex2nemeth supports `pstricks`. So if your pictures are in the form

```
\begin{figure}[ht]
\begin{pspicture}(-2,-2)(2,3)
⟨ps picture commands⟩
\end{pspicture}
\caption{A picture}\label{mypic1}
\end{figure}
```

The program, while running, will:

- move the contents from `\begin{pspicture}` to `\end{pspicture}` to a separate file in your working directory,
- leave a comment in place of the figure to "see figure `label`",
- change all `pstricks` labels to Braille in the new picture files.

So while your file has been transcribed, you now have to modify the picture files it produced to give them proper characteristics for the blind. This part is discussed below with an example.

Assume now that we have a `file.tex` without any pictures in it. We start by simplifying the preamble. We should not have complicated macros. For example, running heads' configuration must be removed. It makes no sense for the blind. Customization of sections, chapter heads, etc., make no sense and must be removed.

Any `\tableofcontents` or similar is also removed; this needs some explanation. Braille files are not in a typeset format such as pdf files. They are simple text files. In order to predict the page of, say, the chapter of a book one needs to know how many lines will be embossed per page and how many braille characters per line. This information is not a standard. Embossers have different settings and it is only the driver of the embosser that could know this information. So a conversion program such as latex2nemeth cannot have access to such information. This is one of the reasons that the output of the program is split into chapters — to give the opportunity to the blind to organize in different folders (or to use tabs) the material of the book.

Latex2nemeth will not parse your `\usepackage` commands but will mostly ignore them. `\newtheorem` and simple `\newcommand` (with or without arguments) are supported. Finally the file must be in UTF-8 encoding. We now start the attempt to convert.

Run `xelatex` or `lualatex` in order to check that your file compiles and produces the `file.aux` file which is needed for the references mechanism.

Now run

```
latex2nemeth file.tex file.aux
```

Most of the time the first run will fail. Typically the user has forgotten to remove visual parts from the preamble. The program will inform you of the line and column of the problem it encountered. Fix it and re-run the above command. After enough corrections of your `.tex` file, the program will succeed. It will produce a `.nemeth` file for each chapter. These are plain text Braille files but in UTF-16 encoding. We need to convert them to UTF-8 and then either import them to LibreOffice for embossing or convert them to LibreOffice automatically.

Let's see the manual procedure first. Conversion to UTF-8 can be done with `iconv`:

```
iconv -f utf-16 -t utf-8 file0.nemeth > file0-u8.txt
```

Now convert to a LibreOffice `.odt` file:

```
libreoffice --headless --convert-to odt file0-u8.txt >/dev/null
```

This will produce `file0-u8.odt`. LibreOffice has a builtin default for the font. But we need a font that has Braille characters, such as DejaVu-Serif. So the final step is to open the `.odt` file, select the whole text (Control-a) and change the font to DejaVu Serif. Save the file.

LibreOffice has a plugin called `odt2braille`. This plugin must be installed in order to be able to drive the embosser. With the plugin installed, open the `odt` file and choose File→Emboss.

The whole process can be automated by a simple script such as this

```
#!/bin/sh
#get a random name first of 8 chars
tmpdir=`cat /dev/urandom | tr -cd 'a-f0-9' | head -c 8`

#make a folder
mkdir $tmpdir

#get the base name of the file to convert
file=`basename "$1" .nemeth`

#convert nemeth from utf16 to utf8
iconv -f utf-16 -t utf-8 "$1" >$file.txt

#convert txt file to odt
libreoffice --headless --convert-to odt $file.txt >/dev/null

# odt is setup with a bultin template for conversions
# from text that uses Liberation Mono font.
# we need DejaVu Serif. We change the font and repack
# the odt file.
unzip -qq -d $tmpdir $file.odt
rm -f $file.odt
find $tmpdir -type f | xargs sed -i 's/Liberation Mono/DejaVu Serif/g'
( cd $tmpdir; zip -qq -r ../$file.odt . )

#cleanup
/bin/rm -rf $tmpdir $file.txt
```

11.1 Conversion of pictures

Now let us turn to pictures. This is most of the work because we have to replace all labels in the picture with Braille (unless you used pstricks in which case the program automatically transcribes the labels) and make new placement decisions, since the Braille is usually long and will not fit in the original position of the label. The easy part is to make the picture lines wider so they can be detected by the hands of the blind. All lines should vary from 1.2 mm minimum to 1.8 mm. We can use this range to distinguish between logically different lines. For example, suppose we want to graph the function $f(x) = x^2$ from -2 to 2. The original graph may look like the one in Figure 1. The Braille for $f(x) = x^2$ is ⠋⠷⠭⠾ ⠨⠅ ⠭⠘⠼⠇ (we will come to this soon). We will change the axis width to 1.2 mm and the graph of the function to 1.8 mm. Since the file will be a pdf file produced in a tactile printer on micro-capsule paper, the Braille is not embossed. So we need to increase its character size to at least 24pt in order to be readable. Moreover the font must be a font such as `NewCMSans10-Book.otf` so that the Braille dots are for blind and not for sighted persons (as is the case with `NewCM10-Book.otf`). So the final graph will be as in Figure 2.

Finally we need an easy way to get the labels into Braille if we used a system other than `pstricks` for our graphics (e.g., `tikz`). An easy way, although time-consuming, is to use a command-line script for this. Create a script, say `l2n.sh`, with contents:

```
#!/bin/bash
echo "\documentclass{article}\usepackage{amsfonts}\begin{document}" \
     > ~/tmp/l2n.tex
echo "$1" >> ~/tmp/l2n.tex
```

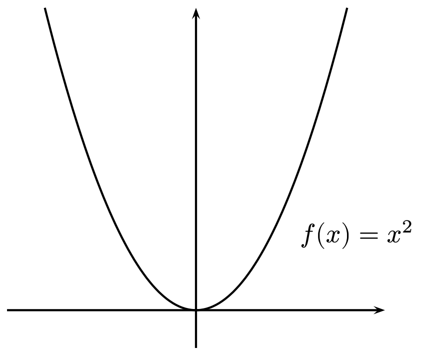


Figure 1: The $f(x) = x^2$ function for sighted persons.

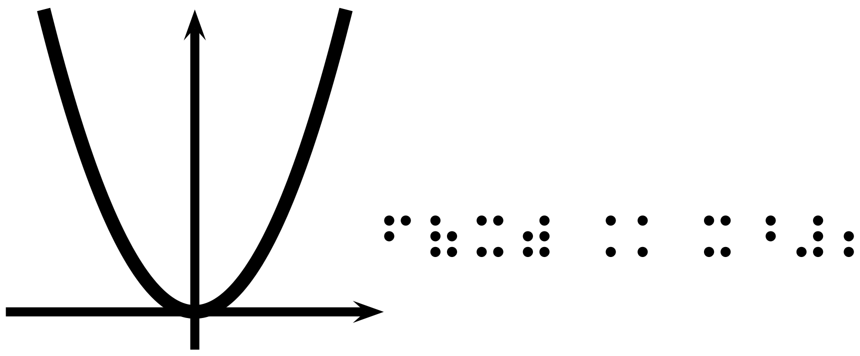


Figure 2: The $f(x) = x^2$ function for the blind

```
echo "\end{document}" >> ~/tmp/l2n.tex
touch ~/tmp/l2n.aux
cd ~/tmp/
latex2nemeth l2n.tex l2n.aux 2>/dev/null
iconv -f utf-16 -t utf8 l2n0.nemeth

echo " "
rm -f l2n.tex l2n0.nemeth l2n.aux
```

Then on the command-line, to get the Braille string for $f(x) = x^2$ run this

```
    sh l2n.sh "\$f(x)=x^2\$"
```

and copy the output to your picture file at the proper label place. Produce the pdf file as you would normally (say with `xelatex`[1] or `lualatex`) and proceed to the tactile printer with micro-capsule paper. Pictures go as pdfs to tactile printers and the Braille text of the TEX files go as `odt` files to embossers.

12 Implementation

Latex2nemeth is written in Java using the JavaCC compiler construction tool. Its design is based on object-oriented techniques such as the Interpreter and Composite design patterns [1] for the representation of mathematical expressions. In order to support spatial aligned structures, as in the case of the `\cfrac` command, a two-dimensional buffer is created for every Braille expression, which is filled in a bottom-up fashion, so as to correctly calculate the dimensions of containing boxes, for example, the width and height of numerator and denominator in a fraction expression. In this way, a generic mechanism for two-dimensional structures was implemented. However, in expressions such as fractions (command `\frac`) which can be ex-

[1] You may need `xelatex-unsafe` if you are using `pstricks`

pressed in Nemeth code in both linear and two-dimensional arrangements, the current version of the program only provides the linear form of the output.

13 Symbols included in `unimath-symbols.pdf` but unsupported in Nemeth

`\arabicmaj`	𞻰	arabic mathematical operator meem with hah with tatweel
`\arabichad`	𞻱	arabic mathematical operator hah with dal
`\inttop`	⌠	top half integral
`\intbottom`	⌡	bottom half integral
`\varhexagonlrbonds`	⌬	six carbon ring, corner down, double bonds lower right etc
`\lparenuend`	⎛	left parenthesis upper hook
`\lparenextender`	⎜	left parenthesis extension
`\lparenlend`	⎝	left parenthesis lower hook
`\rparenuend`	⎞	right parenthesis upper hook
`\rparenextender`	⎟	right parenthesis extension
`\rparenlend`	⎠	right parenthesis lower hook
`\lbrackuend`	⎡	left square bracket upper corner
`\lbrackextender`	⎢	left square bracket extension
`\lbracklend`	⎣	left square bracket lower corner
`\rbrackuend`	⎤	right square bracket upper corner
`\rbrackextender`	⎥	right square bracket extension
`\rbracklend`	⎦	right square bracket lower corner
`\lbraceuend`	⎧	left curly bracket upper hook
`\lbracemid`	⎨	left curly bracket middle piece
`\lbracelend`	⎩	left curly bracket lower hook
`\vbraceextender`	⎪	curly bracket extension
`\rbraceuend`	⎫	right curly bracket upper hook
`\rbracemid`	⎬	right curly bracket middle piece
`\rbracelend`	⎭	right curly bracket lower hook
`\intextender`	⎮	integral extension
`\harrowextender`	⎯	horizontal line extension (used to extend arrows)
`\sumtop`	⎲	summation top
`\sumbottom`	⎳	summation bottom
`\sqrtbottom`	⎷	radical symbol bottom
`\lvboxline`	⎸	left vertical box line

`\rvboxline`	⎹	right vertical box line
`\elinters`	⏧	electrical intersection
`\blocklefthalf`	▌	left half block
`\blockrighthalf`	▐	right half block
`\circlelefthalfblack`	◐	circle, filled left half [harvey ball]
`\circlerighthalfblack`	◑	circle, filled right half
`\circlebottomhalfblack`	◒	circle, filled bottom half
`\circletophalfblack`	◓	circle, filled top half
`\circleurquadblack`	◔	circle with upper right quadrant black
`\blackcircleulquadwhite`	◕	circle with all but upper left quadrant black
`\blacklefthalfcircle`	◖	left half black circle
`\blackrighthalfcircle`	◗	right half black circle
`\invwhiteupperhalfcircle`	◚	upper half inverse white circle
`\invwhitelowerhalfcircle`	◛	lower half inverse white circle
`\ularc`	◜	upper left quadrant circular arc
`\urarc`	◝	upper right quadrant circular arc
`\lrarc`	◞	lower right quadrant circular arc
`\llarc`	◟	lower left quadrant circular arc
`\lrblacktriangle`	◢	lower right triangle, filled
`\llblacktriangle`	◣	lower left triangle, filled
`\ulblacktriangle`	◤	upper left triangle, filled
`\urblacktriangle`	◥	upper right triangle, filled
`\squareleftblack`	◧	square, filled left half
`\squarerightblack`	◨	square, filled right half
`\squareulblack`	◩	square, filled top left corner
`\squarelrblack`	◪	square, filled bottom right corner
`\triangleleftblack`	◭	up-pointing triangle with left half black
`\trianglerightblack`	◮	up-pointing triangle with right half black
`\squareulquad`	◰	white square with upper left quadrant
`\squarellquad`	◱	white square with lower left quadrant
`\squarelrquad`	◲	white square with lower right quadrant
`\squareurquad`	◳	white square with upper right quadrant
`\circleulquad`	◴	white circle with upper left quadrant
`\circlellquad`	◵	white circle with lower left quadrant
`\circlelrquad`	◶	white circle with lower right quadrant
`\circleurquad`	◷	white circle with upper right quadrant
`\ultriangle`	◸	upper left triangle
`\urtriangle`	◹	upper right triangle
`\lltriangle`	◺	lower left triangle

`\lrtriangle`	⊿	lower right triangle
`\quarternote`	♩	music note (sung text sign)
`\eighthnote`	♪	eighth note
`\twonotes`	♫	beamed eighth notes
`\iinfin`	⧜	incomplete infinity
`\laplac`	⧠	square with contoured outline
`\downtriangleleftblack`	⧨	down-pointing triangle with left half black
`\downtrianglerightblack`	⧩	down-pointing triangle with right half black
`\squaretopblack`	⬒	square with top half black
`\squarebotblack`	⬓	square with bottom half black
`\squareurblack`	⬔	square with upper right diagonal half black
`\squarellblack`	⬕	square with lower left diagonal half black
`\diamondleftblack`	⬖	diamond with left half black
`\diamondrightblack`	⬗	diamond with right half black
`\diamondtopblack`	⬘	diamond with top half black
`\diamondbotblack`	⬙	diamond with bottom half black
`\mttzero`	𝟶	mathematical monospace digit 0
`\mttone`	𝟷	mathematical monospace digit 1
`\mtttwo`	𝟸	mathematical monospace digit 2
`\mttthree`	𝟹	mathematical monospace digit 3
`\mttfour`	𝟺	mathematical monospace digit 4
`\mttfive`	𝟻	mathematical monospace digit 5
`\mttsix`	𝟼	mathematical monospace digit 6
`\mttseven`	𝟽	mathematical monospace digit 7
`\mtteight`	𝟾	mathematical monospace digit 8
`\mttnine`	𝟿	mathematical monospace digit 9

References

[1] E. Gamma, R. Helm, et al. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley, Boston, MA, USA, 1994.

◇ Andreas Papasalouros
University of the Aegean
Department of Mathematics
832 00 Karlovassi
Samos, Greece
`http://www.samos.aegean.gr/math/andpapas/cv_en.html`

◇ Antonis Tsolomitis
University of the Aegean
Department of Mathematics
832 00 Karlovassi
Samos, Greece
`http://myria.math.aegean.gr/~atsol`

The Treasure Chest

These are the new packages posted to CTAN (`ctan.org`) from August–October 2022. Descriptions are based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. More information about any package can be found at `ctan.org/pkg/`*pkgname*.

◇ Karl Berry
`https://tug.org/TUGboat/Chest`
`https://ctan.org/topic`

biblio

`ctan-bibdata` in `biblio`
Bibliography of CTAN packages; updated daily.

fonts

`cooperhewitt` in `fonts`
Cross-engine support for the Cooper Hewitt sans serif family, designed for the Smithsonian.

`heros-otf` in `fonts`
Using the OpenType fonts TEX Gyre Heros.

`neo-euler` in `fonts`
OpenType version of Zapf's Euler math fonts.

`pagella-otf` in `fonts`
Using the OpenType fonts TEX Gyre Pagella.

`schola-otf` in `fonts`
Using the OpenType fonts TEX Gyre Schola.

`termes-otf` in `fonts`
Using the OpenType fonts TEX Gyre Termes.

graphics

`chemobabel` in `fonts`
Convert chemical structures from ChemDraw, MDL or SMILES using Open Babel.

`pgf-periodictable` in `graphics/pgf/contrib`
Customizable periodic table of elements.

`wheelchart` in `graphics/pgf/contrib`
Draw wheel charts.

info

`mathtrip` in `info`
Formulae from different fields of mathematics.

macros/latex/contrib

`abntexto` in `macros/latex/contrib`
Support for Associação Brasileira de Normas Técnicas (ABNT) standards.

`abspos` in `macros/latex/contrib`
Absolute placement with coffins.

`colorframed` in `macros/latex/contrib`
Fix color problems with the `framed` package.

`coolfn` in `macros/latex/contrib`
Typeset long legal footnotes.

`darkmode` in `macros/latex/contrib`
Provide `IfDarkMode...` conditionals.

`democodetools` in `macros/latex/contrib`
Show code and its typeset results.

`docshots` in `macros/latex/contrib`
TEX code next to PDF snapshots in `.dtx`.

`eolang` in `macros/latex/contrib`
Formulas and graphs for the EO language.

`gitstatus` in `macros/latex/contrib`
Git information as a watermark or variables.

`huaz` in `macros/latex/contrib`
Automatic Hungarian definite articles.

`jobname-suffix` in `macros/latex/contrib`
Compile differently based on the filename.

`opencolor` in `macros/latex/contrib`
Colors from the Open Color library for UI design.

`photobook` in `macros/latex/contrib`
Support for image-based books.

`ppt-slides` in `macros/latex/contrib`
Slide decks à la PowerPoint.

`se2thesis` in `macros/latex/contrib`
KOMA-Script-based thesis class for Software Engineering II, University of Passau (Germany).

`textcsc` in `macros/latex/contrib`
Typing caps-to-small-caps text.

`udes-genie-these` in `macros/latex/contrib`
Thesis class for the Faculté de génie at the U. de Sherbrooke (Canada).

`ufrgscca` in `macros/latex/contrib`
Support for undergraduates at the Federal U. of Rio Grande do Sul (Brazil), Engineering School.

`unigrazpub` in `macros/latex/contrib`
University of Graz Library Publishing Services.

`wargame` in `macros/latex/contrib`
Preparation of hex and counter wargames.

m/l/c/beamer-contrib/themes

`beamertheme-simpleplus` in `m/l/c/b-c/themes`
A simple and clean theme.

macros/luatex/latex

`japanese-mathformulas` in `macros/luatex/latex`
Compiling basic math formulas in Japanese using LuaLATEX.

`luatruthtable` in `macros/luatex/latex`
Generate boolean truth tables in LuaLATEX.

`piton` in `macros/luatex/latex`
Typeset Python listings without external programs. Written in Lua, using LPEG.

support

`texaccents` in `support`
Convert text accent control sequences to Unicode. Written in Snobol.

`support/texaccents`

Die TEXnische Komödie 3/2022

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`dante.de`). Non-technical items are omitted.

Martin Sievers, Einladung zur digitalen Herbsttagung 2022 [Invitation to the digital autumn meeting 2022]; pp. 7–8

On November 19, 2022, DANTE will have a digital autumn meeting.

Uwe Ziegenhagen, Time to say Goodbye – Stabwechsel im Büro von DANTE e. V. [Time to say goodbye — Staff change in the office of DANTE e. V.]; pp. 9–11

Karin Dornacher retired, Frank Gerhard is her successor.

Robin Garcia, Bericht zur Sommertagung von DANTE e. V. 2022 in Magdeburg [Report on the Dante summer meeting in Magdeburg]; pp. 11–16

DANTE e. V.'s summer meeting took place in Magdeburg, Sachsen-Anhalt.

Adelheid Bonnetsmüller, Having Fun with LATEX: Glücksspiel sauber gesetzt [Having fun with LATEX: Typesetting dice games]; pp. 16–20

A tutorial on typesetting dice.

Adelheid Bonnetsmüller, Having Fun with LATEX: Mein Hut, der hat drei Ecken. . . [Having fun with LATEX: My hat, it has three corners. . .]; pp. 20–21

A tutorial on using the `realhats` package to typeset tiny hats.

William Wenig, Skalierbare Vektorgrafiken einbinden mit Inkscape [Embedding scalable vector graphics with Inkscape]; pp. 21–27

A tutorial on embedding Inkscape graphics.

Henning Hraban Ramm, ConTEXt kurz notiert [ConTEXt short news]; pp. 27–29

News from the ConTEXt world.

Frank Mittelbach, LATEX News: Issue 35, November 2022 [sic]; pp. 30–46

German translation of this LATEX news installment, published in *TUGboat* 43:2, and on the LATEX web site: `latex-project.org/news`.

Jürgen Fenn, Neue Pakete auf CTAN [New packages on CTAN]; pp. 46–50

List of new packages on CTAN.

Johannes Hielscher, Comment to *Druck oder Nichtdruck* in DTK 02/2022 [To print or not to print]; pp. 51–52

A reader's comment on the mentioned article.

[Received from Uwe Ziegenhagen.]

doi.org/10.47397/tb/43-3/tb135komo

La Lettre GUTenberg 46, 2022

La Lettre GUTenberg is a publication of GUTenberg, the French-language TEX user group (`www.gutenberg-asso.fr`).

La Lettre GUTenberg #46

Published October 20, 2022.

Patrick Bideault, Éditorial [Editorial]; pp. 1–5

Patrick Bideault, Les différents travaux de l'association [The various works of the group]; pp. 7–8

Denis Bitouzé, Compte rendu du conseil d'administration [Report of board's meetings]; pp. 9–12

Yvon Henel, L'écureuil sur l'épaule du typographe [The squirrel on the typographer's shoulder]; p. 13

About the association's website.

Patrick Bideault, Maxime Chupin & Yvon Henel, Et maintenant, une bonne *vieille* veille technologique ! [Technology watch]; pp. 14–21

82 new CTAN packages, May–October 2022.

Stefan Kottwitz, Les serveurs de DANTE [DANTE's servers]; pp. 22–25

Translation of the article "Bericht über Projektförderung von LATEX-Servern" published in *Die TEXnische Komödie* 2/2022, pp. 6–10. Stefan Kottwitz is hosting, among others, the French Q&A website `texnique.fr`.

Rémi Angot, Coopmaths, un collectif d'enseignants de mathématiques [Coopmaths, a maths teachers collective]; pp. 26–28

Maxime Chupin, De LATEX vers le braille/nemeth [From LATEX to braille/nemeth]; pp. 28–34

Maxime Chupin, À travers le cyberespace déchaîné : à propos de la somme de deux nombres [Internet report: about the sum of two numbers]; pp. 35–38

Maxime Chupin, La fonte de ce numéro : Bitstream Charter [This issue's font: Bitstream Charter]; pp. 39–43

Patrick Bideault & Maxime Chupin, En bref [At a glance]; pp. 44–47

Short news about the packages `sillypage` and `profcollege`, about TUG sponsoring GUTenberg's DOI project, about the new website dedicated to GUTenberg's journals (`publications.gutenberg-asso.fr`) and more.

[Received from Patrick Bideault.]

doi.org/10.47397/tb/43-3/tb135lettre

2023 TEX Users Group election

TUG Elections committee

The terms of TUG President and five TUG Directors will expire as of the 2023 Annual Meeting, expected to be held in July or August 2023. Three positions are open; thus eight are to be filled.

The terms of these directors will expire in 2023:
Barbara Beeton, Paulo Cereda, Ulrike Fischer,
Jim Hefferon, Norbert Preining.

Continuing directors, with terms ending in 2025:
Karl Berry, Johannes Braams, Kaja Christiansen,
Klaus Höppner, Frank Mittelbach, Ross Moore,
Arthur Rosendahl.

The election to choose the new President and Directors will be held in early Spring of 2023. Nominations for these openings are now invited. A nomination form is available on this page or via `tug.org/election`.

The TUG Bylaws provide that "Any member may be nominated for election to the office of TUG President/ to the Board by submitting a nomination petition in accordance with the TUG Election Procedures. Election ... shall be by ... ballot of the entire membership, carried out in accordance with those same Procedures."

The name of any member may be placed in nomination for election to one of the open offices by submission of a petition, signed by two other members in good standing, to the TUG office; the petition and all signatures must be received by the deadline stated below. A candidate's membership dues for 2023 must be paid before the nomination deadline. The term of TUG President is two years, and the term of Director is four years.

A list of informal guidelines for all TUG board members is available at `tug.org/election/guidelines.html`. It describes the basic functioning of the TUG board, including roles for the various offices and ethical considerations. The expectation is that all board members will abide by the spirit of these guidelines.

Requirements for submitting a nomination are listed at the top of the form. The deadline for receipt of completed nomination forms and ballot information is

07:00 a.m. PST, 1 March 2023

at the TUG office in Portland, Oregon, USA. No exceptions will be made. Forms may be submitted by fax, or scanned and submitted by email to `office@tug.org`; receipt will be confirmed by email. In case of any questions about a candidacy, the full TUG Board will be consulted.

Information for obtaining ballot forms from the TUG website will be distributed by email to all members within 21 days after the close of nominations. It will be possible to vote electronically. Members preferring to receive a paper ballot may make arrangements by notifying the TUG office; see address on the form. Marked ballots must be received by the date noted on the ballots.

Ballots will be counted by a disinterested party not affiliated with the TUG organization. The results of the election should be available by mid-April, and will be announced in a future issue of *TUGboat* and through various TEX-related electronic media.

2023 TUG Election — Nomination Form

Eligibility requirements:

- TUG members whose dues for 2023 have been paid.
- Signatures of two (2) members in good standing at the time they sign the nomination form.
- Supplementary material to be included with the form: passport-size photograph, a short biography, and a statement of intent. The biography and statement together may not exceed 400 words.
- Names that cannot be identified from the TUG membership records will not be accepted as valid.

The undersigned TUG members propose the nomination of:

Name of Nominee: ____________________

Signature: ____________________

Date: ____________________

for the position of (check one):

☐ **TUG President**

☐ **Member of the TUG Board of Directors**

for a term beginning with the 2023 Annual Meeting.

1. ____________________
(please print)

____________________ ____________
(signature) (date)

2. ____________________
(please print)

____________________ ____________
(signature) (date)

Return this nomination form to the TUG office via postal mail, fax, or scanned and sent by email. Nomination forms and all required supplementary material (photograph, biography and personal statement for inclusion on the ballot, dues payment) must be received at the TUG office in Portland, Oregon, USA, no later than

07:00 a.m. PST, 1 March 2023.

It is the responsibility of the candidate to ensure that this deadline is met. Under no circumstances will late or incomplete applications be accepted.

Supplementary material may be sent separately from the form, and supporting signatures need not all appear on the same physical form.

☐ 2023 membership dues paid
☐ nomination form
☐ photograph
☐ biography/personal statement

TEX Users Group
Nominations for 2023 Election
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

(email: `office@tug.org`; fax: +1 815 301-3568)

TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `tug.org/consultants.html`. If you'd like to be listed, please see there.

Dangerous Curve
Email: `typesetting (at) dangerouscurve.org`

Typesetting for over 40 years, we have experience in production typography, graphic design, font design, and computer science, to name a few things. One DC co-owner co-authored, designed, and illustrated a TEX book (*TEX for the Impatient*).

We can ▪ convert your documents to LATEX from just about anything ▪ type up your handwritten pages ▪ proofread, copyedit, and structure documents in English ▪ apply publishers' specs ▪ write custom packages and documentation ▪ resize and edit your images for a better aesthetic effect ▪ make your mathematics beautiful ▪ produce commercial-quality tables with optimal column widths for headers and wrapped paragraphs ▪ modify bibliography styles ▪ make images using TEX-related graphic programs ▪ design programmable fonts using METAFONT ▪ and more! (Just ask.)

Our clients include high-end branding and advertising agencies, academics at top universities, leading publishers. We are a member of TUG, and have supported the GNU Project for decades (including working for them). All quote work is complimentary.

Hendrickson, Amy
57 Longwood Avenue Apt. 8
Brookline, MA 02446
+1 617-738-8029
Email: `amyh (at) texnology.com`
Web: `www.texnology.com`

Full time LATEX consultant for more than 30 years; have worked for major publishing companies, leading universities, and scientific journals. Our macro packages are distributed on-line and used by thousands of authors. See our site for many examples: `texnology.com`.

▪ *LATEX Macro Writing:* Packages for books, journals, slides, posters, e-publishing and more; Sophisticated documentation for users.

▪ Design as well as LATEX implementation for e-publishing, print books and journals, or specialized projects.

▪ Data Visualization, database publishing.

▪ Innovative uses for LATEX, creative solutions our speciality.

▪ LATEX Training, customized to your needs, on-site or via Zoom. See `https://texnology.com/train.htm` for sample of course notes.

Call or send email: I'll be glad to discuss your project with you.

Dominici, Massimiliano
Email: `info (at) typotexnica.it`
Web: `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: `www.texnical-designs.com`

LATEX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized LATEX packages and classes to meet your needs. Contact us to discuss your project or visit the website for further details.

LATEX Typesetting
Email: `enquiries (at) latextypesetting.com`
Web: `latextypesetting.com`

LATEX Typesetting has been in business since 2013 and is run by Vel, the developer behind `LaTeXTemplates.com`. The primary focus of the service is on creating high quality LATEX templates and typesetting for business purposes, but individual clients are welcome too.

I pride myself on a strong attention to detail, friendly communication, high code quality with extensive commenting and an understanding of your business needs. I can also help you with automated document production using LATEX. I'm a scientist, designer and software developer, so no matter your field, I've got you covered.

I invite you to review the extensive collection of past work at the Showcase `latextypesetting.com/showcase`. Submit an enquiry for a free quote!

Monsurate, Rajiv
Web: `www.rajivmonsurate.com`
`latexwithstyle.com`
I offer: design of books and journals for print and online layouts with LaTeX and CSS; production of books and journals for any layout with publish-ready PDF, HTML and XML from LaTeX (bypassing any publishers' processes); custom development of LaTeX packages with documentation; copyediting and proofreading for English; training in LaTeX for authors, publishers and typesetters.

I have over two decades of experience in academic publishing, helping authors, publishers and typesetters use LaTeX. I've built typesetting and conversion systems with LaTeX and provided TeX support for a major publisher.

Sofka, Michael
Email: `michael.sofka (at) gmail.com`
Professional TeX and LaTeX consulting and programming services. I offer 30 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in TeX and LaTeX: Automated document conversion; Programming in Perl, Python, C, R and other languages; Writing and customizing macro packages in TeX or LaTeX, `knitr`.

If you have a specialized TeX or LaTeX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

Veytsman, Boris
132 Warbler Ln.
Brisbane, CA 94005
+1 703-915-2406
Email: `borisv (at) lk.net`
Web: `www.borisv.lk.net`
TeX and LaTeX consulting, training, typesetting and seminars. Integration with databases, automated document preparation, custom LaTeX packages, conversions (Word, OpenOffice etc.) and much more.

I have about two decades of experience in TeX and three decades of experience in teaching & training. I have authored more than forty packages on CTAN as well as Perl packages on CPAN and R packages on CRAN, published papers in TeX-related journals, and conducted several workshops on TeX and related subjects. Among my customers have been Google, US Treasury, FAO UN, Israel Journal of Mathematics, Annals of Mathematics, Res Philosophica, Philosophers' Imprint, No Starch Press, US Army Corps of Engineers, ACM, and many others.

We recently expanded our staff and operations to provide copy-editing, cleaning and troubleshooting of TeX manuscripts as well as typesetting of books, papers & journals, including multilingual copy with non-Latin scripts, and more.

Warde, Jake
90 Resaca Ave.
Box 452
Forest Knolls, CA 94933
+1 650-468-1393
Email: `jwarde (at) wardepub.com`
Web: `myprojectnotebook.com`
I have been in academic publishing for 30+ years. I was a Linguistics major at Stanford in the mid-1970s, then started a publishing career. I knew about TeX from editors at Addison-Wesley who were using it to publish beautifully set math and computer science books.

Long story short, I started using LaTeX for exploratory projects (see website above). I have completed typesetting projects for several journal articles. I have also explored the use of multiple languages in documents extensively. I have a strong developmental editing background in STEM subjects. If you need assistance getting your manuscript set in TeX I can help. And if I cannot help I'll let you know right away.

TeXnology Inc.
Amy Hendrickson
57 Longwood Ave. #8
Brookline, MA 02446
+1 617-738-8029
Email: amyh (at) texnology.com
Web: https://texnology.com
Full time LaTeX consultant for more than 30 years; have worked for major publishing companies, leading universities, and scientific journals. Our macro packages are distributed on-line and used by thousands of authors. See our site for many examples: texnology.com.

- **LaTeX Macro Writing:** Packages for books, journals, slides, posters, e-publishing and more; Sophisticated documentation for users.
- Design as well as LaTeX implementation for e-publishing, print books and journals, or specialized projects.
- Data Visualization, database publishing.
- Innovative uses for LaTeX, creative solutions our speciality.
- LaTeX Training, customized to your needs, on-site or via Zoom. See https://texnology.com/train.htm for sample of course notes.

Call or send email: I'll be glad to discuss your project with you.

Calendar

2022

Nov 5	ISType 2022, “Mukaddeme”, Arabic Script Typography, Istanbul, Turkey. `istype.com/2022`
Nov 12	GUTenberg extraordinary meeting to consider and adopt new bylaws (online). `gutenberg-asso.fr/Assemblee-generale-extraordinaire-refonte-des-statuts-12-novembre-2022`
Nov 18 – 19	TypoDay, “Typography for Children”, Hosted online by IDC School of Design (IDC), Indian Institute of Technology Bombay. `www.typoday.in`
Nov 19	DANTE 2022 Herbsttagung (online), `www.dante.de/veranstaltungen/herbst2022`
Nov 19	TeXConf 2022 (Japan; online) `texconf2022.tumblr.com`
Dec 6 – 9	SIGGRAPH Asia 2022, Daegu, South Korea. `sa2022.siggraph.org`
Dec 11	GUTenberg annual meeting (online). `gutenberg-asso.fr/Assemblee-generale-du-11-decembre-2022`

2023

Mar 1	**TUG election:** nominations due, 07:00 a.m. PST. `tug.org/election`
Mar 24	*TUGboat* **44**:1, submission deadline.
May	BachoTEX 2023, 28th BachoTEX Conference, Bachotek, Poland. `www.gust.org.pl/bachotex/2022-eno`
May 10 – 13	Association Typographique Internationale, ATypI Paris 2023 (hybrid) Paris, France. `atypi.org/conferences-events/atypi-paris-2023`
Jun 26 – 29	SHARP 2023, “Affordances and Interfaces: Textual Interaction Past, Present and Future”, Society for the History of Authorship, Reading & Publishing. Hosted online by the University of Otago, New Zealand. `www.sharpweb.org/main/conferences`
Jun 28 – 30	Twenty-first International Conference on New Directions in the Humanities, “Literary Landscapes: Forms of Knowledge in the Humanities”, Sorbonne University, Paris, France. `thehumanities.com/2023-conference`
Jul 10 – 14	Digital Humanities 2023, Alliance of Digital Humanities Organizations, “Collaboration as Opportunity”, Graz, Austria. `dh2023.adho.org`
Jul 31 – Aug 4	Balisage: The Markup Conference (virtual). `www.balisage.net`
Aug 21 – 25	23rd ACM Symposium on Document Engineering, Limerick, Ireland. `doceng.org/doceng2023`
Sep 5	The Updike Prize for Student Type Design, application deadline, 5:00 p.m. EST. Providence Public Library, Providence, Rhode Island. `prov.pub/updikeprize`
Sep 10 – 16	17th International ConTEXt Meeting, Prague–Sibřina, Czech Republic. `meeting.contextgarden.net/2023`

Owing to the COVID-19 pandemic, schedules may change. Check the websites for details.

Status as of 1 November 2022

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 43 (2022), No. 3