TUGboat

Volume 39, Number 1 / 2018

TEX Users Group

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

Individual memberships

2018 dues for individual members are as follows:

- 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 also have the option to receive *TUGboat* and other benefits electronically, at a discount.

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

Journal subscriptions

TUGboat subscriptions (non-voting) are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate for 2018 is $110.

Institutional memberships

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

Trademarks

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

[printing date: May 2018]

Printed in U.S.A.

Addresses

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

Telephone

+1 503 223-9994

Fax

+1 815 301-3568

Web

tug.org
tug.org/TUGboat

Electronic Mail

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

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

Technical support for TEX users:
support@tug.org

Contact the Board of Directors:
board@tug.org

Mathematics can make typesetting better, and
beautiful typesetting makes mathematics *much* better

Geoffrey Shallit
All I Really Needed to Know
I Learned from Donald Knuth,
slides, Knuth80, January 8–10, Piteå

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 39, NUMBER 1, 2018
PORTLAND, OREGON, U.S.A.

TUGboat editorial information
This regular issue (Vol. 39, No. 1) is the first issue of the 2018 volume year. The second issue is planned to be the proceedings of the annual conference, and the third another regular issue.

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 up to one year after print publication, 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 submission deadline for the first 2018 issue is March 16.

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

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

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

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

TUG Institutional Members

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

American Mathematical Society, *Providence*, *Rhode Island*

Association for Computing Machinery, *New York*, *New York*

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

CSTUG, *Praha*, *Czech Republic*

Harris Space and Intelligence Systems, *Melbourne*, *Floida*

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

Maluhy & Co., *São Paulo*, *Brazil*

Marquette University, *Milwaukee*, *Wisconsin*

Masaryk University, Faculty of Informatics, *Brno*, *Czech Republic*

MOSEK ApS, *Copenhagen*, *Denmark*

Nagwa Limited, *Windsor*, *UK*

New York University, Academic Computing Facility, *New York*, *New York*

Overleaf, *London*, *UK*

ShareLaTeX, *United Kingdom*

Springer-Verlag Heidelberg, *Heidelberg*, *Germany*

StackExchange, *New York City*, *New York*

Stockholm University, Department of Mathematics, *Stockholm*, *Sweden*

TEXFolio, *Trivandrum*, *India*

TNQ, *Chennai*, *India*

University College, Cork, Computer Centre, *Cork*, *Ireland*

Université Laval, *Ste-Foy*, *Québec*, *Canada*

University of Ontario, Institute of Technology, *Oshawa*, *Ontario*, *Canada*

University of Oslo, Institute of Informatics, *Blindern*, *Oslo*, *Norway*

From the president

Boris Veytsman

2018 is promising to be an interesting year for us. The upcoming conferences: April 28–May 2 is BachoTEX in Bachotek, Poland; June 25–27 is PracTEX in Troy, NY, USA; July 20–22 is TUG18 in Rio de Janeiro, Brazil; and September 2–8 is the 12th ConTEXt meeting in Prague-Sibřina. Please do not procrastinate and take advantage of the early bird registration discounts! I would also like to mention that the TUG Annual General Meeting will be held at TUG18.

Broadly speaking, there are two kinds of conferences in science and technology: field-oriented and tool-oriented. The difference is between a conference on breast cancer (how can we tackle this disease?) and a conference on MRI applications (what problems can we solve using this device?). While our conferences are mostly tool-oriented (we have a great tool — TEX!), they have also always attracted talks about things only tangentially related to TEX, but interesting in themselves: fonts, typography, printing history, and many others. The last TUG meeting, at Bachotek, had a bookbinding workshop and a talk about musical notation in old paintings, accompanied by a concert of recovered music. Thus our conferences become somewhat field-oriented, the field being fine & scientific typography in the modern world. I think it is a good feature. So if you have an interesting talk in mind, but are in doubt whether it is TEX-related, consider submitting it anyway.

On the software front, TEX Live 2018 is moving up its schedule this year. Hopefully it will be released before this issue of *TUGboat* is finished. Also, the TEX Development fund made its first award for 2018, for LuaTEX support for the `bidi` package, by Vafa Khalighi. Development fund applications are always welcome; see `tug.org/tc/devfund`.

We are now discussing ways to help with funding the TEX accessibility project, which will make the creation of tagged PDFs easier. This is a very important development for blind and visually impaired scientists and engineers. It is also necessary if we want TEX to stay relevant: many governments are going to require compliance with accessibility standards from publishers: either those who receive government money, or, eventually, all major publishers. This means that anybody who uses TEX in professional activities is interested in making TEX-produced output compliant. Thus we as stewards of TEX must find ways to fund and steer this development effort.

One of the ways to do this is to organize a consortium of publishers and other stakeholders such as the Unicode consortium and PDF association. This organization, affiliated with TUG, would raise funds for the development *and* steer the development. If this is the path to take, we need to write down the charter of the new organization, establish the dues levels for institutional and individual membership and set up the structure of the consortium. If you want to help with this work, or have contacts among potential stakeholders, please write me or the Board (`president@tug.org` or `board@tug.org` respectively). We also have a mailing list for these discussions, `lists.tug.org/accessibility`, and a Web page with links to relevant standards and papers, `tug.org/twg/accessibility`.

Another project important for the relevance of TEX into the future is an educational initiative. We have some interested teachers on our mailing list, `lists.tug.org/edutex`, and have collected information about lesson plans and courses at `tug.org/twg/edutex`. Among the ideas discussed on the list has been the organization of summer TEX camps for school students. If you want to take part in this or any education-related activities, please get in touch!

Of course, these projects, conferences, publications require money. We are grateful to all members and donors who make this work possible with their dues and generous donations. Our ranks include both individual and institutional members; I'm glad to welcome a new institutional member: Harris Space and Intelligence Systems.

We started several experiments to increase the ranks of individual members as well. First, there is a trial option (new users only): a year of full TUG membership for just $20. We offer it at a loss: these dues are lower than the physical cost of shipping *TUGboat* and the TEX Collection DVD. If you choose this option and decide TUG membership is valuable enough, please consider a (tax deductible in the US) donation. And, of course, we hope you will renew in 2019. Second, all new members in 2018 will get a letterpress-printed postcard *and* will participate (unless declined) in a lottery for a book by Jerry Kelly and Martin Hutner, *A Century for the Century: Fine Printed Books from 1900 to 1999.*

So, if you are a member and want to help either by volunteering or by donating — or if you are not yet a member, but are TUG-curious — we are always glad to welcome any help!

Happy TEXing!

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

Editorial comments

Barbara Beeton

Birthdays

Donald E. Knuth, 10 January 1938

In celebration of Don's 80th birthday, Knuth80, in two parts: a conference and an organ concert, was held in Piteå, Sweden. The concert was the premiere of Don's composition *Fantasia Apocalyptica*, an interpretation in music and video of the book of Revelation.

An adapted version of the talk by Yannis Haralambous, one of the invited speakers, appears later in this issue. The program and other highlights of the celebration are online at `knuth80.elfbrink.se`.

Gudrun Zapf von Hesse, 2 January 1918

It took 70 years for it to happen: Gudrun Zapf's first typeface — Hesse Antiqua — was released in digital form on her 100th birthday.

This typeface was not originally intended for print, but instead to be used for stamping title lettering on leather book covers and spines. Her first skilled craft was bookbinding, to which was added lettering, punchcutting, and typeface design. The story of the transformation of the design to digital type is told by the craftsman who accomplished it, Ferdinand Ulrich, at `fontshop.com/content/hesse-antiqua`. What a wonderful 100th birthday present!

Staszek Wawrykiewicz, RIP

On February 7, the TEX world lost a staunch supporter. Staszek, a founding member of GUST, the Polish TEX users group, was a well loved attendee at BachoTEX as well as an active member of the TEX Live team.

Staszek was a dependable and welcome presence at BachoTEX and other TEX meetings in Poland — I first met him at EuroTEX'94 in Gdańsk. An avid musician, he and his guitar could be counted on to lead the other participants in song around the bonfire. He will be greatly missed.

A personal remembrance by Norbert Preining appears later in this issue.

Goodbye Glisterings, hello Duckboat

As he announced in our previous issue, Peter Wilson is retiring from his position as compiler of the Glisterings column, after a run of 17 years. (The first of the series appeared in *TUGboat* **22**:4, in 2001.) We have been treated to an abundance of useful TEXniques, ideas for making documents more visually attractive, and pithy sayings. Let's take this opportunity to thank Peter for his contribution, and wish him well as he continues TEXing at his Herries Press.

Actually, Peter would like to share a few last words. His column has always included the wording "... hopefully not making things worse through any errors of mine." In the last column there was a bit about using the `changepage` package (section 2.2, All is not what it seems). He apologises, but the start of this example code should have included "`\strictpagecheck`" like this:

```
\usepackage{changepage}
\strictpagecheck
...
```

For more information about this command please read the `changepage` manual (`texdoc changepage`).

Appearing for the first time, also in our previous issue, is the new column, the Duckboat, by Professor Paulinho van Duck, co-conspirator with Carla Maggi. Inspired by the plethora of interesting questions and answers at `tex.stackexchange.com`, and taking its sub-theme from that forum's inordinate fondness for ducks,[1] the column will carry on the *TUGboat* tradition of collecting and sharing useful TEXtual tidbits, using `tex.sx` as its source, and providing hints on how one can make best use of that resource.

A new "dual" typeface: visible and touchable

An announcement of an interesting new typeface has appeared on the web, although it hasn't been made available to TEX, and it's not clear that it would be possible to do so.

This typeface — Braille Neue — merges a visible alphabet with Braille, and is intended for use in signage. The designer, Kosuke Takahashi, began with the Braille dots, which cannot be moved, and shaped the letters of the Latin alphabet around them. (He first attempted to overlay Japanese characters, but this proved incompatible owing to the complex character shapes.) He hopes that Braille Neue will be used somewhere at the 2020 Tokyo Olympics and Paralympics.

Images of the font are shown on the designer's website (`kosuke.tk/work-rattt.html`) and in an article about the project.[2]

[1] The `tikzducks` package, new to TEX Live this year, provides ample evidence of this syndrome. If you have any doubt, look at `codegolf.stackexchange.com/questions/159567/blue-duck-red-duck-gray-duck/159718#159718` for a demonstration.

[2] `fastcodesign.com/90166173/this-new-typeface-merges-braille-you-can-touch-with-letters-you-can-see`

40 years ago...

I've recently received the newsletter from the Museum of Printing, in Haverhill, Massachusetts. This issue unfolds into a strip 13 inches tall by 40 inches wide, and contains a timeline of print history. The oldest entry (3100 BCE) reads

> Cuneiform, one of the earliest known writing systems developed in Sumer (modern day Iraq). Wedge-shaped marks were made on clay tablets by a blunt stylus cut from a reed.

(Peter Wilson, in his keynote at TUG 2007 in San Diego, passed around some exhibits, one of which was a cuneiform tablet. This method of recording text is possibly the most durable, as observed in the title page quote for *TUGboat* **33**:1.)

But the year 1978 is the one with the most appeal to the present audience:

> Last *New York Times* set by Linotype; featured in documentary film Farewell, Etaoin Shrdlu.
>
> TeX typesetting system developed by Donald Knuth. It revolutionized the composition and publication of technical books and journals.
>
> Friends of Museum of Printing founded in Massachusetts.

Many other years mark interesting events, but none quite as notable to us as this one.

This timeline is similar, but not identical, to one that appears on a page of the American Printing History Association's website.[3]

Hyphens, UK style

British hyphenation practice has been "evolving", at least according to Oxford University Press, whose most recent spelling dictionary shows considerable variation from the 1986 edition used to develop the patterns now in use by (LA)TeX. A spirited discussion has been taking place on the `tex-hyphen` list,[4] along with correspondence arising from the effort to try to bring the patterns into current practice. What a kerfuffle (a delightful word)!

Dominik Wujastyk, who was involved in creating the original patterns, is spearheading the effort. Here's hoping that it succeeds.

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

[3] `printinghistory.org/timeline/`
[4] `lists.tug.org/tex-hyphen`

Hyphenation exception log

Barbara Beeton

This is the periodic update of the list of words that TeX fails to hyphenate properly for U.S. English. The full list last appeared in *TUGboat* 16:1, starting on page 12, with periodic updates in *TUGboat*, most recently in 36:1, p. 7.

In the list below, the first column gives results from plain TeX's `\showhyphens{...}`. The entries in the second column are suitable for inclusion in a `\hyphenation{...}` list.

In most instances, inflected forms are not shown for nouns and verbs; note that all forms must be specified in a `\hyphenation{...}` list if they occur in your document. The full list of exceptions, as a TeX-readable file, and the scripts used to create it, appears at `https://ctan.org/pkg/hyphenex`.

Like the full list, this update is in two parts: English words, and names and non-English words that occur in English texts.

Thanks to all who have submitted entries for the list. As a reminder of one of the idiosyncrasies of TeX's hyphenation algorithm: hyphens will not be inserted before the number of letters specified by `\lefthyphenmin`, nor after the number of letters specified by `\righthyphenmin`. For U.S. English, `\lefthyphenmin=2` and `\righthyphenmin=3`; thus no word shorter than five letters will be hyphenated. (For the details, see *The TeXbook*, page 454.) This particular rule is violated in some of the words listed; however, if a word is hyphenated correctly by TeX except for "missing" hyphens at the beginning or end, it has not been included here.

Some other permissible hyphens have been omitted for reasons of style or clarity. While this is at least partly a matter of personal taste, an author should think of the reader when deciding whether or not to permit just one more breakpoint in some obscure or confusing word. There really are times when a bit of rewriting is preferable.

One other warning: Some words can be more than one part of speech, depending on context, and have different hyphenations; for example, 'analyses' can be either a verb or a plural noun. If such a word appears in this list, hyphens are shown only for the portions of the word that would be hyphenated in the same way regardless of usage.

The reference used to check these hyphenations is *Webster's Third New International Dictionary*, unabridged.

The list — English words

adamant	ad-a-mant
analect	an-a-lect
anonymity	an-o-nym-i-ty
anony-mous(ly)	anon-y-mous(ly)
an-tibi-otic	anti-bi-ot-ic
an-tipodes	an-tip-o-des
awestruck	awe-struck
backpedal(s,ing)	back-pedal(s,-ing)
bioin-for-mat-ics	bio-in-for-mat-ics
biomass	bio-mass
chameleon	cha-me-leon
chaotic	cha-ot-ic
cognoscenti	co-gno-scen-ti
columbine	col-um-bine
colum-nar	col-um-nar
con-tractable	con-tract-able
cor-ti-cos-teroid	cor-ti-co-steroid
cuisines	cui-sines
cus-tomer	cus-tom-er
democ-racy	de-moc-ra-cy
demo-crat	dem-o-crat
demo-cratic	dem-o-crat-ic
demon-strabl(e,y)	de-mon-stra-bl(e,y)
demon-strate	dem-on-strate
demon-strat-ing	dem-on-stra-ting
demon-stra-tion	dem-on-stra-tion
di-alect	dia-lect
di-alec-tal	di-a-lec-tal
di-alec-ti-cal	di-a-lec-ti-cal
di-atom	di-a-tom
di-atoma-ceous	di-a-to-ma-ceous
dilemma	di-lem-ma
dis-tribut-ing	dis-trib-ut-ing
disu-til-ity	dis-util-ity
endgame	end-game
ex-plicit	ex-pli-cit
ex-plic-itly	ex-pli-cit-ly
fidu-ciary(ies)	fi-du-ciary(-ies)
fontenc	font-enc
fragility	fra-gil-i-ty
freesia	free-sia
het-eroin-t-er-face	het-ero-in-ter-face
homonym	hom-o-nym
homonymic	hom-o-nym-ic
homony-mous	ho-mon-y-mous
homonymy	ho-mon-y-my
ho-mo-phone	ho-mo-phone
ho-mo-phonic	ho-mo-phonic
ho-mophonous	ho-moph-o-nous
ho-mophony	ho-moph-o-ny
id-io-gram	id-io-gram
id-i-olect	id-i-o-lect
in-signif-i-cant	in-sig-nif-i-cant
in-tractable	in-trac-ta-ble
in-tractabil-ity	in-trac-ta-bil-ity
ju-nior	jun-ior
labyrinth	lab-y-rinth
labyrinthian	lab-y-rin-thi-an
labyrinthine	lab-y-rin-thine
ma-l-in-vest-ment	mal-invest-ment
men-sch	mensch
nu-cle-osyn-the-sis	nu-cleo-syn-the-sis
pe-nal-ize	pen-al-ize
perispom-ena(on)	peri-spome-na(on)
physics	phys-ics
polypep-tide	poly-pep-tide
predilec-tion	predi-lec-tion
pseudon-um-ber	pseu-do-num-ber
ragged	rag-ged
roundish	round-ish
roundish-ness	round-ish-ness
runnable	run-nable
scalar	sca-lar
SIu-nits	SI-units
spec-troscopy	spec-tros-co-py
stretch-a-bil-ity	stretch-abil-ity
sub-s-e-lect	sub-se-lect
sub-s-e-lected	sub-se-lected
sub-s-e-lec-tion	sub-se-lec-tion
su-perel-lipse	super-el-lipse
su-perel-lip-ti-cal	super-ellip-ti-cal
su-perel-lip-ti-cally	super-ellip-ti-cally
su-perel-lip-ti-cal-ness	super-ellip-ti-cal-ness
supraor-di-nate	su-pra-ordi-nate
syn-chronic-ity	syn-chro-ni-city
syn-onym	syn-o-nym
syn-ony-mous	syn-on-y-mous
syn-onymy	syn-on-y-my
tableau	tab-leau
ther-moe-las-tic-ity	ther-mo-elas-tic-ity
ther-mome-ter	ther-mom-eter
ther-monu-clear	ther-mo-nu-clear
ti-tanate	ti-ta-nate
tractable	trac-ta-ble
uber-men-sch	uber-mensch
un-pre-dictabl(e,y)	un-pre-dict-a-bl(e,y)

Names and non-English words used in English text

Alexan-der	Alex-an-der
Alexan-drine	Alex-an-drine
Bigelow	Big-elow
Bringhurst	Bring-hurst
Carnegie	Car-ne-gie
Columbian	Co-lum-bi-an
com-para-i-son	com-pa-rai-son
Ge-of-frey	Geof-frey
Har-alam-bous	Hara-lam-bous
Knuthian	Knuth-ian
Kun-stakademie	Kunst-aka-de-mie
Mesopotamia	Mes-o-po-ta-mia
Mo-jca	Moj-ca
QW-ERTY	QWERTY
Rochester	Ro-ches-ter
Sin-ga-pore	Singa-pore
Sin-ga-porean	Singa-po-re-an
Wikipedia	Wiki-pe-dia

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

In memoriam: Staszek Wawrykiewicz (1953–2018)

Norbert Preining

We have lost a dear member of our community, Staszek Wawrykiewicz. I got notice that our friend died in an accident the other day. My heart stopped for an instant when I read the news, it cannot be — one of the most friendly, open, heart-warming friends has passed.

Staszek was an active member of the Polish TEX community, and an incredibly valuable TEX Live team member. His insistence and perseverance have saved TEX Live from many disasters and bugs. Although I have been in contact with Staszek over the TEX Live mailing lists for many years, I met him in person for the first time at my first ever BachoTEX, the EuroBachoTEX 2007. His friendliness, openness to all new things, his inquisitiveness, all took a great place in my heart.

I dearly remember the evenings with Staszek and our Polish friends, in one of the Bachotek huts or around the bonfire, him playing the guitar and singing traditional and not-so-traditional Polish music, inviting everyone to join and enjoy together. Rarely have technical and social abilities found such a nice combination as in Staszek.

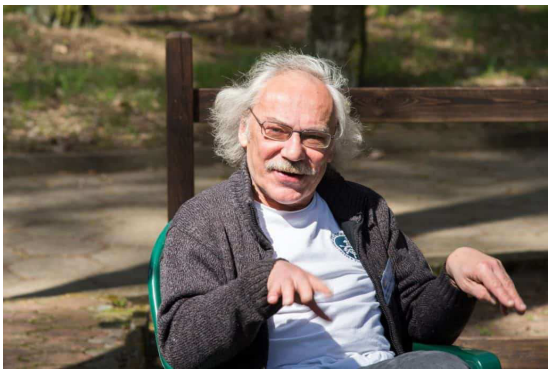
Despite his age he often felt like someone in his twenties, always ready for a joke, always ready to party, always ready to have fun. It is this kind of attitude I would like to carry with me when I get older. Thanks for giving me a great example.

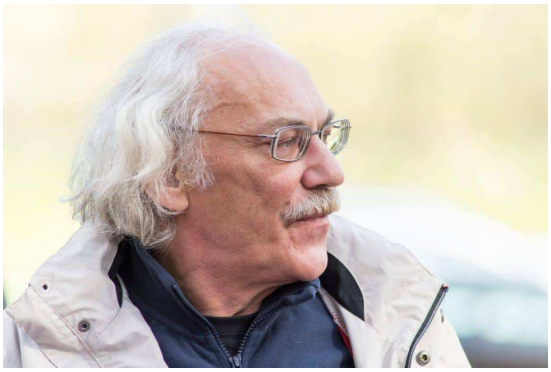
The few times I managed to come to BachoTEX from far Japan, Staszek was always welcoming — it is the feeling of close friends that even if you haven't seen each other for long time, the moment you meet it feels like it was just yesterday. And wherever you go during a BachoTEX conference, his traces and humor were always present.

It is a very sad loss for all of those who knew Staszek. If I could I would like to board the plane just now and join the final service to a great man, a great friend.

Staszek, we will miss you. BachoTEX will miss you, TEX Live will miss you, I will miss you badly. A good friend has passed away. May you rest in peace.

◇ Norbert Preining
Komatsu, Ishikawa, Japan
https://www.preining.info/





[Editor's note: Originally written as a blog post: preining.info/blog/2018/02/in-memoriam-staszek-wawrykiewicz. More about Staszek, in Polish, is on the Polish Wikipedia: pl.wikipedia.org/wiki/Stanis%C5%82aw_Wawrykiewicz.

Photos are courtesy Andrzej Odyniec, released by him to the public domain.]

TEX as a Path, a Talk Given at Donald Knuth's 80th Birthday Celebration Symposium

Yannis Haralambous

Donald E. Knuth's 80th birthday celebration on January 10th, 2018, in Piteå (northern Sweden), was organized as a double event: (a) a scientific symposium[1] where colleagues, former students and friends were invited to give presentations in the fields of algorithmics and combinatorics, and (b) the world première of his multimedia work *Fantasia Apocalyptica* for pipe organ and three video tracks.

The symposium was the opportunity to attend talks by renowned scientists, such as, to mention just a few (in alphabetical order): Persi Diaconis (Stanford), Ron Graham (UC San Diego), Dick Karp (UC Berkeley), Bob Sedgewick (Princeton), Bob Tarjan (Princeton), Andrew Yao (Tsinghua University), ... I didn't count the exact number of Turing prizes, but there must have been four or five, at least.

Fantasia Apocalyptica was a stimulating and intense experience: during an hour and a half we listened to Don's "program" music played on one of the best pipe organs in the world, while (1) reading the unabridged text of John's Revelation in Greek and English, (2) reading the score as it was played and (3) looking at specially drawn Duane Bibby drawings ... the whole in perfect synchronization,[2] and in a beautiful wooden concert room: the *Acusticum* of Piteå's Higher School of Music and Dance.

For this unique event, I was invited to give a talk about Don's work in typography. A vast subject, which I had to fit in only 30 minutes. I spent the six months of preparation of this talk constantly alternating between visionary joy ("I will at last be able to express my gratitude and admiration to Don!") and paralyzing anxiety ("will I ever find something to say that Don doesn't know already?!?"). After the symposium, I wanted to share the result with *TUGboat* readers, through this paper.[3] What follows is an edited and corrected transcription of my talk, with several additional explanations and references.

1 TEX as a Path

It is a big honor for me to give this talk about TEX and typography in front of Don, because TEX literally changed my life. The main keyword of this talk is 'gratitude', even though the word doesn't appear explicitly in it.

Duane Bibby, the famous creator of the lion and lioness characters, has kindly prepared a drawing especially for this occasion:

In it you can see the lion of TEX together with the lioness of METAFONT and the coffee-drinking happy humanoid computer, walking on a path paved by bits. The path starts from a temple with Don's initials and heads to the setting sun. Our two beloved leonine creatures are dwelling on this path and smiling at us, while we read on a road sign: "Happy 80th birthday Don."

You may be wondering why I chose to talk about paths? Well, a path is, of course (Merriam–Webster) *a trodden way, a track specially constructed for a particular use* — which is also the case of TEX — but also, most importantly, *a way of life, a way of conduct, a way of thought.*

There is this famous poem by Cavafy [4, p. 36]:

Σὰ βγεῖς στὸν πηγαιμὸ γιὰ τὴν Ἰθάκη,
νὰ εὔχεσαι νᾶναι μακρὺς ὁ δρόμος,
γεμᾶτος περιπέτειες, γεμᾶτος γνώσεις.

[1] I like the word 'symposium' because it stems from the Greek word *symposion*, meaning "drinking together"!

[2] Experiencing *Fantasia Apocalyptica* reminded me of literate programming and specifically of reading volume E of *Computers & Typesetting*: the volume on Computer Modern fonts where you have annotated glyph images, METAFONT code and comments on facing pages. Of course in the case of *Fantasia* the experience was much stronger since the act of reading the Biblical text(s) and the score had to follow the pace of music, while (a different?) part of the brain was cognitively processing the incoming music. It is characteristic of Don to provide the experiencer with many simultaneous levels and modalities of information — so much information, indeed, that selecting a small part of it that your mind can humanly capture and process, is a creative process per se.

[3] The slides are on the symposium's Web site `http://knuth80.elfbrink.se/talks/` and you can also watch the talk on YouTube: `https://youtu.be/P1AxaFQzfT4`.

When you set out on the journey to Ithaca,
pray that road be long,
full of adventures, full of knowledge.

where the "path", the "way of life", is given a name: 'Ithaca'. Don had many Ithacas in his life and he attained them all, but according to Cavafy, what is important is not attaining the destination but rather the journey, a journey full of adventures, full of knowledge.

2 The Outline of the Talk

This is the moment where the speaker traditionally gives the outline of eir[4] talk. This was a difficult task for me because the first idea of an outline that came to my mind was this:

Photo credit: Zabair Khan, CC BY-NC 2.0.

It is the Grand Canyon. I remember somebody saying once that "talking about TEX is like trying to describe the Grand Canyon in a postcard". You may wonder "Why the Grand Canyon?" It happens that when you look at TEX you sometimes feel like contemplating pure beauty:

[4] I'm using Michael Spivak's gender-neutral pronouns [21, p. xv].

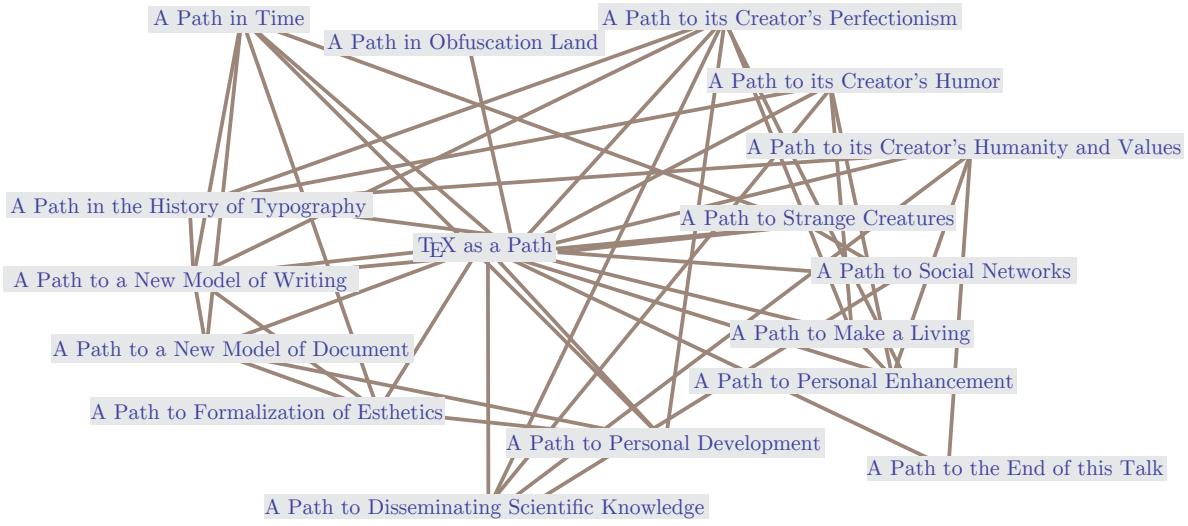
Photo credit: Helari Hellenurm, CC BY-NC-ND 2.0.

and sometimes you feel rather like rafting in a heavy stream:

Photo credit: Andrew Peacock, used by permission.

which is another aspect of the Grand Canyon, and of TEX. . .

In fact, my real outline is shown in the figure at the bottom of the page, namely a graph whose vertices are the various topics I want to address, and edges represent semantic relatedness above a given threshold between the topics. Once you have



this semantic graph, a narrative thread can be obtained algorithmically as a shortest Hamiltonian path (Hamiltonian since we want to visit all vertices exactly once, and shortest since we want to keep the talk as coherent as possible). In fact, instead of applying shortest Hamiltonian algorithms I did something easier: I detected communities of vertices (the sections of this talk) and I drew a path joining them in the most sensible way.

3 History

First of all, TEX is a path in time. You may say that it starts in 1942 with a 4-year old Milwaukee boy [18, p. 1] going to the local library to read "Babar the King", and ending up in the local newspaper as the youngest member of the local Book Worm Club. But actually the first real milestone was the first book ever typeset in TEX: "Lena Bernice" [9, 15], typeset in 1978 and printed in 100 copies. And then of course, another important milestone was *The TEXbook* (1984), as the main entrance to TEX for most of us old-timers, and finally, in 1990, Don's decision to freeze TEX [17].

TEX is also a path in the history of typography. We all know that Gutenberg modeled writing by inventing movable metal type-based typography. Don has dematerialized Gutenberg's model using boxes and glue. Also, many people in the Renaissance attempted to mathematically model letter shapes, Pacioli being one of them. Don, once again, has provided a solution to the problem by devising the programming language METAFONT. In a previous talk [20] Martin Ruckert has extensively talked about the letter 'S' and the difficulties this letter caused for Don (see also [14] and [12]). The creative path from Gutenberg and Pacioli to Don is a well-known one, but that's not all. In fact, we can consider that TEX and METAFONT are much more than that since Don's work has introduced two new models: a *model of writing* and a *model of documents*.

4 A Model of Writing

In 1968, in their foundational work "The Sound Pattern of English" [5], Chomsky and Halle describe a language's phonological structure — be it English or any other language — through what they call "sound rules". For them, word forms we use in speech can be obtained from "abstract forms", after some (mostly standard) transformations. For example, the French article <le> /lə/ (note that we use angle brackets for the written word and slashes for pronunciation represented in the IPA) becomes <les> /le/ in the plural, so if we look only at the surface there is /lə/ for the singular and /le/ for the plural. But when the plural form is followed by a noun starting with a vowel, as in <les ans> /lezɑ̃/, a /z/ sound appears out of nowhere (in French this is called *liaison*). Chomsky and Halle say that there is an "abstract sound" /z/ which corresponds to the plural suffix and the sound rule will send it to nothing (= will keep it mute) unless a noun starting with a vowel follows the article.

This will come as no surprise since that "abstract plural suffix" has existed in the French language for many centuries, well before Chomsky and Halle. But it exists not in speech but in the written modality, where it is represented by an <s>.

We can schematize Chomsky and Halle's approach by:

$$\text{abstract sounds} \xrightarrow{\text{sound rules}} \text{pronunciation.}$$

What is of interest to computer scientists is that these "sound rules" are in fact *production rules of a context-sensitive formal grammar* (and this was the very reason why Chomsky introduced the formal grammar concept in the first place).

Richard Sproat [22], inspired by Anneke Nunn [19], has extended this to *graphemes*. Graphemes are the basic elements of writing in the same way that phonemes are the basic elements of spoken language. You can define graphemes by the method of minimal pairs: if two "drawings" in the same context give rise to different semantics, then they are different graphemes. For example, every English reader recognizes the words <hat> and <cat> as having different semantics, therefore <h> and <c> are different English language graphemes. On the other hand, <cɑt> and <cat> represent the same semantics for the average reader of English, so <ɑ> and <a> represent the same grapheme of English language; we call them *allographs*.

Sproat [22] proposes the following diagram:

$$\begin{array}{ccc} \text{abstract sounds} & \xrightarrow{\text{sound rules}} & \text{pronunciation} \\ \Big\downarrow{\scriptstyle \text{phoneme to grapheme conversion}} & & \\ \text{abstract spelling} & \xrightarrow{\text{autonomous spelling rules}} & \text{spelling,} \end{array}$$

where all arrows represent rules that can be described by means of regular languages. In the case of graphemes the regular language is of a special kind; he calls it a *planar* language and it has not one but *five* concatenation operators, corresponding to five relative placements of character pairs: "over" $\overset{\uparrow}{\cdot}$, "under" $\overset{\downarrow}{\cdot}$, "on the left" $\overset{\leftarrow}{\cdot}$, "on the right" $\overset{\rightarrow}{\cdot}$, and "surrounding" $\odot$. So, for example, the Chinese character <鱗> can be described as the formal word <魚$\overset{\rightarrow}{\cdot}$[米$\overset{\downarrow}{\cdot}$[夕$\overset{\rightarrow}{\cdot}$𠂤]]> if we consider the "components"

appearing in this decomposition as the alphabet of our formal language (notice that we also need brackets since this kind of planar concatenation is not associative).

It is time now to return to TEX. The reader may find a strong similarity between graphemes and Unicode characters. Let's understand well that there is an important difference: the former are language-specific (since they are defined by minimal pairs in a *given* language) and the latter strive for universality. But we can also consider a grapheme as an equivalence class of drawings, and a Unicode character as an equivalence class of glyphs. It is also noteworthy that Sproat's five concatenation operators have their analogs (even though only intended for Chinese Han characters) in Unicode: the *Ideographic description characters* `U+2ff0-2ffb`, which are not 5 but 12. And still, in my humble opinion, these are not enough: consider for example the Vietnamese acute accent which has to sit *on the right side* of the circumflex accent: this is much subtler than simply saying that the former "lies above" the latter. It should be clear by now that some precise way of describing grapheme/Unicode character interaction is still needed. I claim that

Claim 1. *The ideal tool for describing grapheme interaction is TEX!*

Indeed, our beloved TEX, besides being a program and a programming language, is also an algorithmic transformation from tokens (tokens can represent abstract phonemes or abstract graphemes) into DVI command sequences (which again can represent graphemes, as well as their geometric interactions).

One may argue that to obtain DVI one needs also fonts (or, at least, font metric information). But don't you always need them? When we want to precisely describe the interaction between two graphemes, we need information on their shapes.

And this raises a second issue: when studying a (written) language at some point you need to describe its glyphs/allographs. And since these can vary, while representing the same character/grapheme, you need some *flexible* way of describing them. You need a *meta*-description. I claim that

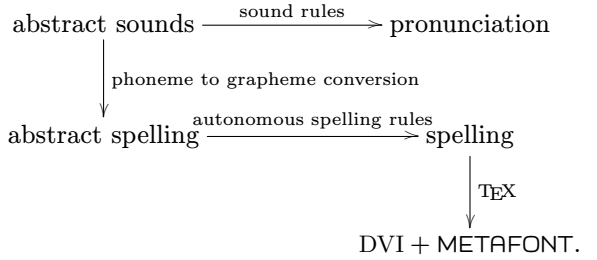
Claim 2. *METAFONT is the ideal tool for describing glyphs (a.k.a. allographs) and equivalence classes of glyphs (a.k.a. graphemes).*

Take for example the description of the Euro symbol as given in [7]. This description uses precise geometrical properties but takes into account neither stroke width variation, nor special techniques for diagonal junction lightening,[5] which are necessary for real-world typography. METAFONT can easily model the precise equations of [7], and still allow for metaness and typographic performance.

To resume, I claim that

Claim 3. *TEX+METAFONT can be considered as a very efficient new model of the written form of (any) language, whether for studying it (as does linguistics) or to produce contents in it.*

And hence I propose an extension of the diagram above to:



5 A Document Model

Conventional document models, such as XML (with XSL-FO and SVG) or PDF, or the many word processor file formats, contain a mixture of characters and glyphs, the former mostly from the user's input and the latter after processing and applying properties. For example, XML contains only characters, but SVG allows the user to describe glyphs (in a very rudimentary way) and to use them in an XSL-FO context. PDF contains only glyphs, but when the user goes through the GUI for operations such as copying or searching, PDF reader software will either use the font encoding to obtain characters corresponding to glyphs or a special command called `ActualText` [1, §14.9.4] to attach a Unicode string to some part of the document, for example to a sequence of glyphs. A document in one of these formats contains a static textual content, either in glyphs or in characters, or in some weak correspondence between the two.

TEX has introduced a new, infinitely more powerful document model, by providing not only the source and the result but also *the complete process* of document creation. In TEX you know when every transformation is applied and for what reason. Take for example the <fl> ligature and the French <œ> digraph. In the DVI file they are both single glyphs, but if you look at the TEX process you will realize that <œ> was there from the beginning (either as a Unicode character in TEX versions supporting Unicode, or as an `\oe` macro), while <fl> has appeared in the node list when the typesetting process met the

[5] See the *apex_corr* and *notch_cut* parameters in Computer Modern fonts.

font and got information from it about availability (and necessity) of the ligature in the specific font. And, of course, in a German setting where the letters <f> and <l> belong to different morphemes — such as in the word <Auflage> — the ligature will not be applied.

Accessing the whole process from source to output gives you the five Ws: who, when, why, what and how of every part of your document. To give an analogy, suppose you have to talk about a great (wo)man: you will talk not only about what e left, but about eir entire life. Life is a transformation of projects into memories — take this conference: six months before, it was a project, a dream; then it became an event; six months later, it will become a memory. You don't want to keep only memories, you want to access the whole of it. In our analogy, TEX and METAFONT source is your project, DVI+PK is your memory, what is important is *how* you obtain it. A similar analogy is music: you can have a score (the source) and you can have a recording (the result). But the most important part is neither of them, it is the music making process per se.

For all these reasons I claim that

Claim 4. *TEX and METAFONT can be used to describe the input of a document, its output, as well as the complete process of obtaining the output from the input. This is a new way to model documents, and certainly the most powerful I can imagine.*

6 TEX and Esthetics

When Gutenberg printed his Bible, he actually committed a fraud: he sold Bibles to wealthy monks pretending they were handwritten, and got paid accordingly. Therefore the esthetic of that first printed book was rather the one of manuscripts of its time. But once printing technology emigrated to Italy, immensely talented artists such as Aldus Manutius or Francesco Griffo created a new esthetic for the printed book. This esthetic canon has evolved during centuries but has never really been formalized. There have been manuals and methods, but never a formal specification.

Don has done this formal specification algorithmically and included the algorithm into TEX. When he defined badness, demerits, penalties and the like [16], he was in fact formalizing notions and rules that have existed for centuries as part of a craftsman's skills. And speaking about TEX innards, here is another opportunity to mention a path: the solution to the problem of the most pleasant paragraph is indeed a shortest path in a directed acyclic graph, and TEX managed to be operational in very small systems (as were operating systems in the late seventies) precisely because the algorithm for finding a shortest path in a directed acyclic graph is of no more than linear complexity.

7 Disseminating Scientific Knowledge

After having discussed what a marvelous achievement TEX has been as part of the history of writing, of documents and of typographical esthetics, let us turn to the impact of TEX on humans, be it universally, or individually, or for specific communities.

Firt of all, let us give some figures:

- There are currently 1,344,162 papers on the `arXiv.org` archive, out of which 1,230,793 (that is 92%) are written in TEX[6];
- for those needing help, there are 147,645 brilliantly and multiply *answered questions* on the `tex.stackexchange.com` platform[7];
- for those needing extra power, there are 5,411 LATEX packages on CTAN written by 2,472 authors[8];
- the collaborative online LATEX platform Overleaf has 2,169,037 accounts and hosts the astonishing number of 17,256,112 TEX documents[9];
- *all* presentations of the Knuth80 symposium were prepared in TEX![10]

These figures show that, thanks to TEX, nowadays *anyone* with access to a computer can *freely* and *efficiently* produce scientific and scholarly documents with high quality presentation standards.

8 TEX's Impact on Individuals

When you use TEX you become humble, patient, hardened.

Debugging a TEX document is experiencing

δι' ἐλέου καὶ φόβου περαίνουσα
τὴν τῶν τοιούτων παθημάτων κάθαρσιν

through pity and fear effecting
the proper purgation of such emotions.

This quote is part of Aristotle's definition of tragedy [3]. TEX users experiencing *κάθαρσις* (*cathartic emotions*) do it because the result is *worth the pain*. There is beauty emerging from the innards of the

[6] As of January 5, 2018 (personal communication with Jim Entwood).

[7] As of January 5, 2018 (information available on the Web site).

[8] As of November 1st, 2017 (personal communication with Gerd Neugebauer).

[9] As of January 6, 2018 (personal communication with John Hammersley).

[10] This is notoriously and shamelessly FAKE NEWS as the reader visiting the symposium's Web site can easily verify by downloading the PDF files of the talks. But all other figures in this paragraph are entirely true!

machine. And the important point is that the TEX user can *always* obtain the result e needs, because e has total control on eir tools. This enhances quality of life, and provides confidence, self-esteem and dignity. (And not frustration like other programs, which are fatally bound by limits.)

Besides personal *development* TEX also leads to personal *enhancement*. Take for example the Preface of the *METAFONTbook*, where Don gives a warning:

> WARNING: Type design can be hazardous to your other interests. Once you get hooked, you will develop intense feelings about letterforms; the medium will intrude on the messages that you read. And you will perpetually be thinking of improvements to the fonts that you see everywhere, especially those of your own design.

I.e., if you read that book you will *change*, you will *evolve*. And indeed, being exposed to TEX and METAFONT extensively alters your senses and produces extra sensitivity to *form*. And this extra sensitivity to form enhances understanding of *content* since it is through form that we access content.

Meanwhile, TEXers are easily recognizable by the common symptoms they present:

- facial spasms when witnessing club lines;
- nightmares populated by overfull boxes;
- a smile when discovering other people's typographical blunders, such as mixing up 'ß' and 'β', or 'ς' (number) and 'ς' (letter), or using an ASCII apostrophe '' instead of a typographic apostrophe '' in text, or omitting an 'œ' digraph in French words like 'cœur';
- euphoria when in presence of a beautiful font;
- visionary ecstasy when in front of an Aldine Press volume;
- olfactory excitation when smelling paper and ink;
- a mood of "home sweet home" when unexpectedly recognizing Computer Modern fonts in a library or bookstore.

There are also people using TEX to earn a living and feed themselves and their families *working at it with all their heart* [Col. 3.23]. Again TEX is the ideal tool for this purpose because it gives the professional user total control of eir typesetting activities and document aspects. E don't have to learn it anew on every new version (since there is no new version) but e can capitalize on eir knowledge.

And then, you have communities of people, people who share their experiences and pleasure, but above all, their creativity. In TEX user groups (or {TEX Users}, as Michael Spivak [21, App. H] calls them) there are neither social, nor racial, nor geographic, nor gender-related, nor age-related, nor computer proficiency-related barriers. TEX-induced friendships last a lifetime.

9 Small Mystery Interlude

TEX may lead you to the encounter of strange creatures like this one:

or this one:

or this one:

These creatures live in tribes:

يجب أن تكون الآلة التي يذبح بها أطول من عنق رقبة الحيوان المذبوح حتى إذا جر بها لا يدخل رأسها فى وسط الذبح لئلا تنكسر في وقت الرد بها على جلد المذبوح ويكون لها عرض وتخن حتى لا تثني وتضطرب حيث يذبح بها ولا يكون فيها اعوجاج ولا ثلم ولا حروشة جارية على خط مستقيم ماضية حادة في غاية ما يكون. ولا تكون زجاجاً ولا خزقا ولا خشباً ولا عظماً ولا قصبا ولا ظفراً لأن كل هذه الأشياء لا يؤمن تفتيت بعضها وتتليمها عند الجر بها على الجلد وانها تكون حديداً حتى يكون الذبح مأمونا بها لأن الحديد أقوى

and their tribes sometimes meet:

يجب أن يكون الضابط للبهيمة غير الذابح كبيرة كانت أو صغيرة ويرمى الذابح يده على عنق المذبوح ويرد الخرزة الكبيرة إلى ما يلي الرأس ويخنق قدامه بالابهام والذى يلى وما بينها وينزل ويذبح ويكون مستقبلا بالمذبوح إلى القبلة القدوسية تمثيلا بالذبائح المقربة كما قال:

וְשָׁחַט אֹתוֹ עַל יֶרֶךְ הַמִּזְבֵּחַ צָפֹנָה לִפְנֵי יהוה (ויקרא א:יא) [וְיִשְׁחַט אֶת הַפָּר לִפְנֵי יהוה (اللاويين ١ الآية ١١) عَلَى جَانِبِ الْمَذْبَحِ إِلَى الشِّمَالِ أَمَامَ الرَبِّ]

and when their tribes meet then the *characters become messengers of peace and hope among humans.*

10 TEX in Obfuscation Land

TEX is also the ideal tool for obfuscation. In a previous talk [6], Erik Demaine was referring to songs

with small complexity [13, p. 20]. Here is one of those songs ("On the first day of Christmas, my true love gave me...") written in obfuscated TEX code by David Carlisle [2]:

```
\let~\catcode~‘76~‘A13~‘F1~‘j00~‘P2jdefA71F~‘7113jdefPALLF
PA’’FwPA;;FPAZZFLaLPA//71F71iPAHHFLPAzzFenPASSFthP;A$$FevP
A@@FfPARR717273F737271P;ADDFRgniPAWW71FPATTFvePA**FstRsamP
AGGFRruoPAqq71.72.F717271PAYY7172F727171PA??Fi*LmPA&&71jfi
Fjfi71PAVVFjbigskipRPWGAUU71727374 75,76Fjpar71727375Djifx
:76jelse&U76jfiPLAKK7172F7117271PAXX71FVLnOSeL71SLRyadR@oL
RrhC?yLRurtKFeLPFovPgaTLtReRomL;PABB71 72,73:Fjif.73.jelse
B73:jfiXF71PU71 72,73:PWs;AMM71F71diPAJJFRdriPAQQFRsreLPAI
I71Fo71dPA!!FRgiePBt’el@ lTLqdrYmu.Q.,Ke;vz vzLqpip.Q.,tz;
;Lql.IrsZ.eap,qn.i. i.eLlMaesLdRcna,;!;h htLqm.MRasZ.ilk,%
s$;z zLqs’.ansZ.Ymi,/sx ;LYegseZRyal,@i;@ TLRlogdLrDsW,@;G
LcYlaDLbJsW,SWXJW ree @rzchLhzsW,;WERcesInW qt.’oL.Rtrul;e
doTsW,Wk;Rri@stW aHAHHFndZPpqar.tridgeLinZpe.LtYer.W,:jbye
```

This is indeed very efficient obfuscation and it works because TEX has a primitive command called `\catcode` which operates in a "transfigurative" way and changes the essence of any character. TEX can follow these transfigurations seamlessly while the human mind is totally unable to read past the first line (at least that is *my* case).

11 TEX's Creator

The path of this talk has allowed us to wander through many aspects of TEX. We left the most important for last: TEX's creator.

C&T, a.k.a. *Computers and Typesetting*, a five-volume book of 2,704 pages, is a *closed universe*. The books have a logical structure, they are produced by programs which have their own logical structure (the two structures being different according to the rules of literate programming). The programs are described by text, which has paragraphs, lines and characters. Characters are also described by programs, which again are described by text.

Take a long breath and think of the complexity of this *œuvre*.

And then consider the fact that even the tiniest character needs the full power of the system. To obtain a humble little comma you need all of METAFONT and TEX, as in oriental theories of the universe [11]:

> *Every atom reflects the whole Universe. How can it be otherwise, since every atom is Primordial Substance?*

Every comma reflects the whole of TEX and METAFONT. So you have a whole universe in front of you, and you can delve inside it. In its center you will encounter its creator.

12 Don's Humor

Don's humor is proverbial, irreverent, unrestrainable and ubiquitous.

Take the following sentences from *The TEXbook*. The first one appears very early, in the Preface:

> *Another noteworthy characteristic of this book is that it doesn't always tell the truth.*

This is hardly expected in a serious book by one of the greatest computer scientists of the 20th century. But it gets even better when the reader arrives at the very last exercise:

> *Final exercise: Find all of the lies in this book, and all of the jokes.*

Then there are Duane Bibby's beautiful, maliciously funny and funnily malicious drawings like this one, from the "Dirty tricks" chapter:

And last, but not least, *The TEXbook* is full of awesome quotations, like the following (one of my favorites):

> *... according to legend, an RCA Marketing Manager received a phone call from a disturbed customer. His 301 had just hyphenated "God."* [10]

13 Don's Humanity and Values

When we read Don's books we *learn things*. But we also *learn how to learn*, and by learning how to learn, we *learn how to teach*, how to make the dullest subject interesting and noteworthy, and that, in return, provides us with a feeling of gratifying creativity. Don cares about every single reader whether e is an expert or a novice, and this is a permanent source of inspiration for us.

The following quote is not from Don but is in *The TEXbook* and represents for me the very *essence of teaching*:

> *Pretend that you are explaining the subject to a friend on a long walk in the woods.* [8]

14 Conclusion

With this "long walk in the woods" (which is yet another path), we arrive at the end of this talk.

Cavafy's *Ithaca* asserts that the voyage is more important than the destination:

> *Ἴθακη σ' ἔδωσε τὸ ὡραῖο ταξείδι.*
> *Χωρὶς αὐτὴν δὲν θἄβγαινες στὸν δρόμο.*
>
> *Ithaca gave you the wondrous voyage:*
> *without her you'd never have set out.*

Without Don, without his Ithacas, we wouldn't have TEX, we would not have this wondrous voyage, without him we would not have set out.

So on this occasion I would like to wish Don a Happy Eightieth Birthday, and solemnly tell him, in the name of all TEX users, past, present and yet to come,

Thank you Don, from the bottom of our hearts!

References

[1] Adobe Systems Inc. Document management — Portable Document Format — Part 1: PDF 1.7, 2008. adobe.com/devnet/pdf/pdf_reference.html.

[2] Anonymous. Pearls of TEX programming. *TUGboat*, 26(3):256–263, 2005. tug.org/TUGboat/tb26-3/tb84pearls.pdf.

[3] S. H. Butcher, editor. *The Poetics of Aristotle*. Macmillan and Co., London, 1895.

[4] Constantinos Cavafy. Ithaca. In *The Collected Poems*. Oxford University Press, 2007. Translated by Evangelos Sachperoglou.

[5] Noam Chomsky and Morris Halle. *The Sound Pattern of English*. Harper & Row, 1968.

[6] Erik Demaine. Fun and games meet computer science, 2018. Talk given at Knuth80. knuth80.elfbrink.se/talks/.

[7] European Commission. The design of the euro, 2002. perma.cc/4NEE-HFBQ.

[8] P. R. Halmos et al. *How to Write Mathematics*. American Mathematical Society, 1973.

[9] Elisabeth Ann James. *Lena Bernice: Her Christmas in Wood County, 1895*. Rainshine Press, Columbus, Ohio, 1978. With illustrations by Jill Carter Knuth.

[10] P. E. Justus. There is more to typesetting than setting type. *IEEE Transactions on Professional Communication*, PC-15(1):13–16, 1972.

[11] William Kingsland. *The Physics of the Secret Doctrine*. The Theosophical Publishing Society, London, 1910.

[12] Donald E. Knuth. 32 years of METAFONT. Talk given at the San Francisco Public Library on Sept. 20, 2016. youtu.be/0LR_lBEy7qU.

[13] Donald E. Knuth. The complexity of songs. *SIGACT News*, 9:17–24, 1977.

[14] Donald E. Knuth. The letter S. *The Mathematical Intelligencer*, 2:114–122, 1980.

[15] Donald E. Knuth. TEX incunabula. *TUGboat*, 5(1):4–11, 1984. tug.org/TUGboat/tb05-1/tb09knut.pdf.

[16] Donald E. Knuth. *Computers & Typesetting*. Addison Wesley, 1984–1986. 5 vols.

[17] Donald E. Knuth. The future of TEX and METAFONT. *TUGboat*, 11(4):489, 1990. tug.org/TUGboat/tb11-4/tb30knut.pdf.

[18] Donald E. Knuth. *Digital Typography*, volume 78 of *CSLI Lecture Notes*. CSLI Publications, 1999.

[19] Anneke Nunn. *Dutch Orthography: A Systematic Investigation of the Spelling of Dutch Words*, volume 6 of *LOT International Series*. Holland Academic Graphics, 1998.

[20] Martin Ruckert. Programming as an art, 2018. Talk given at Knuth80. knuth80.elfbrink.se/talks/.

[21] Michael Spivak. *The Joy of TEX. A Gourmet Guide to Typesetting with the AMS-TEX macro package*. American Mathematical Society, Providence, Rhode Island, 2nd edition, 1990.

[22] Richard Sproat. *A Computational Theory of Writing Systems*. Studies in Natural Language Processing. Cambridge University Press, 2006.

◇ Yannis Haralambous
IMT Atlantique, UMR CNRS 6285 Lab-STICC
Technopôle Brest Iroise CS 83818, 29238 Brest Cedex 3, France
yannis.haralambous (at) imt-atlantique dot fr

TUG is TEX users helping each other

Jonathan Fine

1 What is a user group?

If you are reading this, then you are most likely a TEX user, and a member of the TEX Users Group. The essence of TUG is TEX users helping each other.

For this to happen, we must ask for and offer help to each other. *TUGboat* articles mostly offer useful information. Online forums allow questions to be asked and answered, sometimes in close to real time.

2 The rise and fall of TUG

TUG was founded around 1982. Membership grew rapidly until about 1992. Since then it has declined, with a clear temporary reversal between 1998 and 2003, and a few minor ups and downs. TUG membership is now at its lowest point since about 1985, about 1/3 its peak.

3 TUG's finances

All figures are in thousands, rounded. In 2016 TUG's income was $104. Running the office cost $75, and producing *TUGboat* $25. Legal fees were $14. There were other expenses. At year's end, there was a loss of $21, reducing TUG's assets to $195.

4 Communication is the essence

The essence of TUG is helpful communication between TEX users. Here, of course, Board members have a special responsibility. I ask the Board to do more to encourage and participate in helpful member-to-member communication.

5 The TUG members mailing list

There's already a mailing list for this: http://tug.org/mailman/listinfo/members. Sadly, it's close to dead. In 2017 there were just 2 threads, and 3 solitary messages. Earlier years are similar.

6 You and the revival of TUG

Without helpful open communication between members, TUG will continue to decline. And with helpful open communication, we'll make the best of what there is. Please, if you're a TUG member, email postmaster@tug.org and ask to be placed on the TUG members list. And then a future will be possible.

◇ Jonathan Fine
Milton Keynes
England
jfine2358@gmail.com

LATEX and Jupyter, TikZ and Vega

Jonathan Fine

1 Then and now

When Don Knuth created TEX in the 1970s and 80s, publishing was mostly on paper. TEX was created to solve the problem of computer typesetting, particularly for technical content. The portable computers, including the mobile phone, have changed publishing. Many people prefer laptop and notebook computers to paper books.

2 Laboratory and scientific notebooks

The great experimental physicist Michael Faraday (1791–1867) kept a lab diary. Today we might do this on a computer, as a private blog, or a scientific notebook, such as Jupyter.

TEX and LATEX solved the problem of typesetting, for printing on paper. Today, Project Jupyter develops "open-source software, open standards, and services for interactive and reproducible computing".

3 Jupyter and LATEX

In many ways, Jupyter is now what LATEX was in the 1980s. It's got a growing and well-funded community, and making steady and rapid progress. It is a major and well-respected force.

4 PGF/TikZ and D3/Vega

PGF/TikZ is a deservedly popular TEX-based technical drawing package. In it, PGF/TikZ is a low-level/high-level language pair.

In the parallel universe of scientific web publishing, D3/Vega is a similar language pair, based not on TEX but on HTML5.

Many would benefit from a bridge between TikZ and Vega, particularly those who want high-quality visualisation in both PDF and interactive HTML5.

5 Further reading (and browsing)

In January 2018 Nature published a Toolbox article *Data visualization tools drive interactivity and reproducibility in online publishing*. The URL is https://www.nature.com/articles/d41586-018-01322-9.

Inspired by this Nature article, I gave a talk at the March 2018 London PyData meetup. The URL is https://jfine2358.github.io/slides/2018-nature-jupyter-altair-vega-binder.html.

◇ Jonathan Fine
Milton Keynes
England
jfine2358@gmail.com

Typographers' Inn

Peter Flynn

Fonts and faces and families

I suppose we've all but given up the unequal struggle to distinguish between a family, a face, and a font. I still use the terms separately, out of force of habit, but some work we were doing recently (see 'XeLaTeX' below) allowed me to identify many good examples. One family I installed recently (following its announcement on `comp.text.tex`) was IBM Plex, which is composed of these faces:

1. Plex Serif
2. Plex Serif ExtraLight
3. Plex Serif Light
4. Plex Serif Medium
5. Plex Serif SemiBold
6. Plex Serif Text
7. Plex Serif Thin
8. Plex Sans
9. Plex Sans ExtraLight
10. Plex Sans Light
11. Plex Sans Medium
12. Plex Sans SemiBold
13. Plex Sans Text
14. Plex Sans Thin
15. Plex Mono
16. Plex Mono ExtraLight
17. Plex Mono Light
18. Plex Mono Medium
19. Plex Mono SemiBold
20. Plex Mono Text
21. Plex Mono Thin

I numbered them on a slide for a training course so that the students could see the seven serif, seven sans, and seven monospace components — and with luck, understand the distinction — before explaining that each one came in the four standard font variants: regular, bold, italic, and bold-italic; making 84 in all.

(Incidentally, Plex looks likely to be an excellent choice for documentation, as it is relatively compact for its large x-height, only about 10% wider than CM. Its overall colour is much darker due to the less marked difference between thick and thin strokes (Figure 1), which improves readability, although in long measures it needs a little more leading.)

What I was trying to convey was that, bearing in mind that there are many larger font families such as Univers or Gotham, picking 'a font' is a much more demanding task than it appears. I have mentioned elsewhere [3, p 95] John Lewis' story about designing examples illustrating the choice of typefaces; delicate little script fonts for cosmetic adverts, classical, formal, respectable roman faces for banks, big chunky sans-serif fonts for engineering, and so on; only to discover after a while that he could 'change the typefaces around at will and with ever increasing effect' [4, p 52].

In display material such as advertising or publicity, pretty much anything goes, because the important thing is the visual impact. But in the three classic document classes (books, articles, reports... maybe four if we include theses) it's usually much more important that the choice of typeface remains unnoticed. It's not that the choice is unimportant, but that it's more subtle than just choosing a 'font' that you like the look of, although that is obviously part of the decision.

Fonts and faces and families

I suppose we've all but given up the unequal struggle to distinguish between a family, a face, and a font. I still use the terms separately, out of force of habit, but some work we were doing recently (see 'XeLaTeX' below) allowed me to identify many good examples. One family I installed recently (following its announcement on `comp.text.tex`) was IBM Plex, which is composed of these faces:

1. IBM Plex Serif
2. IBM Plex Serif ExtraLight

Figure 1: IBM Plex in action

In the days of metal type, even the largest printers would have had only a tiny fraction of the typefaces available to the average computer user today: a designer specifying a face not on hand would have had to cost-justify the rental of the matrices needed to cast it, or (in smaller houses) buying sorts in cases cast to order. Nowadays the choice is vast, and there are hundreds of websites providing a range of methodologies for choosing suitable typefaces for different applications.

Commercial typefaces remain expensive — last time I checked, a full set of Gotham was about €600 per user — but the range and quality of free typefaces grows daily, as the selection available with TeX shows. Even the phrase 'available with TeX' is now becoming less significant as XeLaTeX lets you use all your existing TrueType and OpenType font files as I showed in an earlier column [1]. Perhaps what we need now is a font-selection methodology adaptable to the kind of documents TeX users typically create.

XeLaTeX

Moving our own workflows into XeLaTeX raised a number of questions, as I mentioned above. As with most platform changes, there are pros and cons, but the decision was made on the basis of the 'normal' documents we process — that is, continuous text with the traditional document features I mentioned in the last column [2].

We don't specialize in mathematical work, so we don't have the restrictions of math font choice. What does come up regularly, though, is checking the availability of all the accented letters for Latin-alphabet languages, the range of publishers' symbols, and how closely a given (often free) typeface resembles one (often commercial) selected by a client's previous designer.

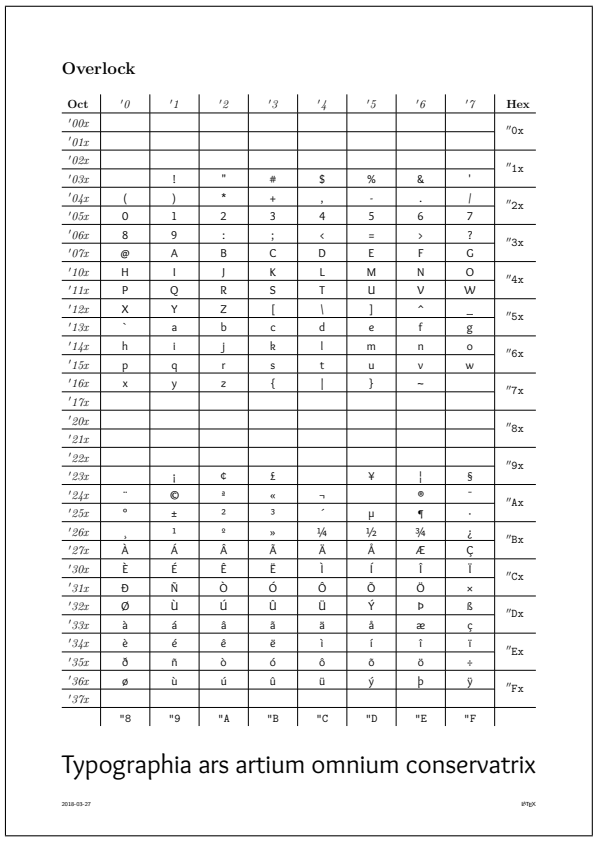


Figure 2: Revised table output for 256-character font display

Many years ago there was a file called `allfnt8.tex` which created an 8×16 grid to display all 128 possible character positions in a font file (at the time). This was later expanded to show a 32-line table for 256 character positions. I still have a copy but I have failed to find it anywhere else, even on CTAN, and it's for Plain TEX only.

Fairly obviously, an upgrade to LATEX code and XHLATEX compatibility was needed, so we have a new version which is being cast as a document class called fontable. It does the same thing as `allfnt8.tex` but it takes the full font name or the `.ttf`/`.otf` file name which XHLATEX uses. It labels the output with the font name, hex and octal character numbers, the sample sentence as shown in Figure 2, and the date, so that users referencing the output can know when it was last tested.

The last step was to remake the font cache as described earlier [1], and then extract all the font names in a form suitable for automating the production of the grids on demand. That revealed a number of things: the size of font families mentioned earlier; the villainous state of the metadata in the font files; and the amount of duplication, at least on our font server.

The duplication was easy to fix, if tedious. We really didn't need six identical copies of Nimbus Mono or Lobster Two — and TEX is by no means the only application to install lots of font files. The metadata is a mess, though: inconsistently-abbreviated font names, with or without spaces, with or without hyphens, sometimes capitalized, sometimes all lowercase or all uppercase, and sometimes even camelCase — and who thought it would be good to name their font `Vietnamese\\040Computer\\040Modern`. The font variants are better: Regular, Medium, Bold, and Italic are fine, but others suffer the same problems as the font names.

It would of course be possible to edit the font binaries and fix the problems, but at the cost of potential incompatibility with code that has been hard-wired to expect or reference the broken strings. It's just something we need to live with: the benefits of using XHLATEX far outweigh the costs.

Afterthought

Has anyone written a web application using the Fontconfig tools, so that users (and clients) can view resident font libraries?

References

[1] Peter Flynn. Typographers' Inn — XHLATEX. *TUGboat*, 37(3):266, Dec 2016. `http://tug.org/TUGboat/tb37-3/tb117inn.pdf`.

[2] Peter Flynn. Typographers' Inn — Layouts. *TUGboat*, 38(1):17, Jan 2017. `http://tug.org/TUGboat/tb38-1/tb118inn.pdf`.

[3] Peter Flynn. Digital Typography. In Kent Norman and Jurek Kirakowski, editors, *Handbook of Human-Computer Interaction*, pages 89–108. Wiley, Hoboken, NJ, Jan 2018.

[4] John Lewis. *Typography: Basic principles: Influences and trends since the 19th century.* Studio Books, London, Jan 1963.

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

Type designer Nina Stössinger speaks at 3rd Annual Updike Prize event

David Walden

The Providence Public Library (PPL) held its 3rd third annual Updike Prize for Student Type Design[1] award ceremony on October 23, 2017. This year's finalists, announced by PPL Special Collections Librarian Jordan Goffin, were Joseph Allegro for his Meadows typeface and Erica Carras for her Raleigh Condensed typeface, with Carras being the winner. The three judges were New England graphic and type designers. The regional Paperworks company provided cash prizes. Carras also received a trophy made from a composing stick.

The actual award ceremony is a small part of the event and came between a presentation by a guest speaker, who is a professional from the type design world, and a question-and-answer session between the audience and the guest presenter. This year's guest presentation was by Nina Stössinger.[2]

The title of Stössinger's presentation was "Looking & Making & Questioning", those words being the subtitles for the three parts of her presentation.

Looking. Stössinger showed lots of photographs of all sorts of different letters seen in everyday life — on the sides of trucks, on buildings, on posters, in store windows, etc., and in all stages of freshness from just printed/painted to seriously decayed, and across the spectrum from formal to vernacular. She is always looking at lettering as she goes through life (she showed a photo of a sign in the Providence train station taken as she was just arriving that day in Providence from New York City). What she sees may help her in unforeseen ways for future type design projects. She recommends looking.

Making. Stössinger told how she spent three years designing her FF Ernestine typeface.[3] Wondering what to do next, she avoided starting the long process to design and perfect another typeface and instead spent several months doing daily type design exercises. She took these exercises from the website `typecooker.com`, drew them by hand, and published them daily online. She explored design decisions quickly, resisting polishing her drawings; and she didn't draw words but rather experimented with odd or useful combinations of letters. She recommends cultivating such making.

Questioning. Stössinger compared typefaces that have the vertical parts of letters being wider than the horizontal parts of them, as is common with Latin fonts, with typefaces that have the horizontal parts wider than the vertical parts, as in Hebrew letters. She noted that Latin letters in some "Wanted" or circus posters have wider horizontal parts, but this is not typical for letters for everyday reading. She set for herself the project of creating a "serif face with stressed horizontals" that was "nice to read", wondering "could this work". The result was her Nordvest typeface as used here:[4]

Rainfrogs

Of type design, Stössinger noted that "some rules really make sense" while "others are just conventions". She encouraged questioning — "think of something new".

Nina Stössinger is employed by Frere-Jones Type of Brooklyn where she also programs "scripts" to help with the repetition in type design. Some of these have developed into tools which she has posted at `github.com/ninastoessinger`.

Prior type designers who have spoken in the PPL's Updike Prize series have been Matthew Carter (who spoke at the time the competition was kicked off, a year before the first prize was awarded),[5] Tobias Frere-Jones (who spoke at the first award ceremony),[6] and Fiona Ross (who spoke at the second award ceremony).[7] Stössinger is perhaps the least well established of these speakers, but her abundant enthusiasm was surely inspirational to the student type designers in the audience.

A video of the 2017 Updike Prize event including Nina Stössinger's presentation should eventually be posted at `youtube.com/user/provlib`.

"Updike" in the prize name comes from the PPL's Daniel Berkeley Updike Collection on the History Printing.[8] Updike, founder of the renowned Merrymount Press and author of the classic *Printing Types: Their History, Forms and Use*, over many years encouraged the PPL to acquire books and other historic materials and gave the collection much material of his own. To qualify for the Updike Prize, student competitors must visit the Updike collection at least once, in additional to submitting their new type design and an essay about it.

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

[1] `provlib.org/updikeprize`
[2] `ninastoessinger.com`
[3] `ernestinefont.com`
[4] `ninastoessinger.com/typefaces/nordvest/`
[5] `tug.org/TUGboat/tb35-1/tb109beet.pdf`
[6] `tug.org/TUGboat/tb36-1/tb112beet.pdf`, p. 4
[7] `tug.org/TUGboat/tb37-3/tb117beet.pdf`, p. 257
[8] `provlib.org/exhibitions/daniel-berkeley-updike-collection-history-printing`

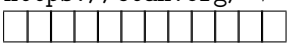
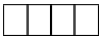
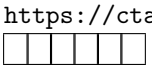
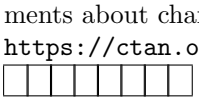
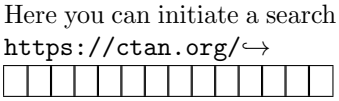
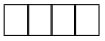
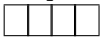
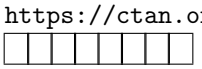
CTAN Quiz

Gerd Neugebauer

Abstract

The portal of the 'Comprehensive TEX Archive Network' (CTAN) contains a multitude of web pages. In this quiz some features and pages are described. As solution the associated url has to be completed and entered into the prepared boxes. The number of boxes can serve as an additional hint. Remember, a url may contain not only letters but also some other special characters.

You can understand this quiz as a chance to explore the depth of the CTAN portal. It can be fun to scan the pages in search of the described features.

1. On this page you can filter the topics cloud:
   `https://ctan.org/↪`
   
2. Here the lion is wagging its tail:
   `https://ctan.org/↪`
   
3. On this page a fanfare horn is blown (in a proper browser and with activated speakers):
   `https://ctan.org/↪`
   
4. On this page the supported browsers are listed:
   `https://ctan.org/↪`
   
5. Here you can see on a map of the world:
   `https://ctan.org/↪`
   
6. On this page you can register a mirror server:
   `https://ctan.org/↪`
   
7. On this page packages are marching past:
   `https://ctan.org/↪`
   
8. Here you can find authors sorted by number of contributed packages:
   `https://ctan.org/↪`
   
9. On this page you can find the latest announcements about changes of packages:
   `https://ctan.org/↪`
   
10. Here you can initiate a search targeted at topics:
    `https://ctan.org/↪`
    
11. This page allows you to share your package with the world:
    `https://ctan.org/↪`
    
12. Here you can find a description for uploading of packages with JSON:
    `https://ctan.org/↪`
    
13. Here you can give the CTAN portal a wooden appearance:
    `https://ctan.org/↪`
    
14. On this page all the packages of Donald Knuth are listed:
    `https://ctan.org/↪`
    
15. There you can find a description for querying CTAN with JSON:
    `https://ctan.org/↪`
    
16. Here you arrive at the complete directory tree of the TEX archive:
    `https://ctan.org/↪`
    
17. Here you can see your own package ratings and more:
    `https://ctan.org/↪`
    
18. On this page you can see how others liked the portal:
    `https://ctan.org/↪`
    
19. Here you can see which TEX user groups you can join and how to contact them:
    `https://ctan.org/↪`
    
20. On this page, brand new packages are listed which have not even been published yet in the archive:
    `https://ctan.org/↪`
    

Answers can be found on page 27.

◇ Gerd Neugebauer
Im Lerchelsböhl 5
64521 Groß-Gerau (Germany)
gene (at) gerd-neugebauer (dot) de
www.gerd-neugebauer.de

The DuckBoat — News from TEX.SE: The Morse code of Ti*k*Z

Herr Professor Paulinho van Duck

Abstract

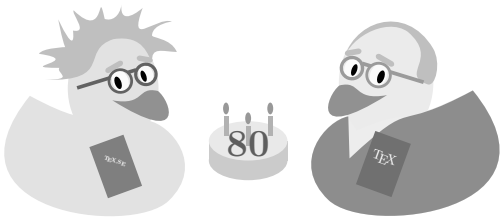
For this installment, Prof. van Duck would like to tell you how the *duck mania* began and infected many TEX.SE users. In the second Quack Guide, you will find a beginner's approach to Ti*k*Z, a powerful package to draw your graphics directly in LATEX.

1 Here I am, again!

Hi, (LA)TEX friends!

If you missed the last issue, I am Prof. van Duck, and I enjoy helping beginners like me!

First of all, let me celebrate a very significant date, and thank our Jedi Master Prof. Knuth for creating the best typesetting system in the world!



Happy Birthday, Prof. DEK!

Secondly, I would like to thank all the TEX.SE friends who warmly appreciated the first DuckBoat. Some of them also used a link to it in their comments, to explain to new users how to ask.

Peter Wilson himself wrote to me suggesting a new topic (something like: *How to add code and images to TEX.SE posts?*), which will be treated in one of the next Quack Guides.

Prof. Enrico Gregorio was so kind to take me with him on stage during his talk at last October's conference of the Italian TEX User Group, the annual GUIT *meeting* (I thank him also for his editing and suggestions about this article).

By the way:

```
\begin{advertising}
```

Are you an Italian (LA)TEX user?

Join the GUIT!

http://www.guitex.org

```
\end{advertising}
```

I am very proud of all that, quack!

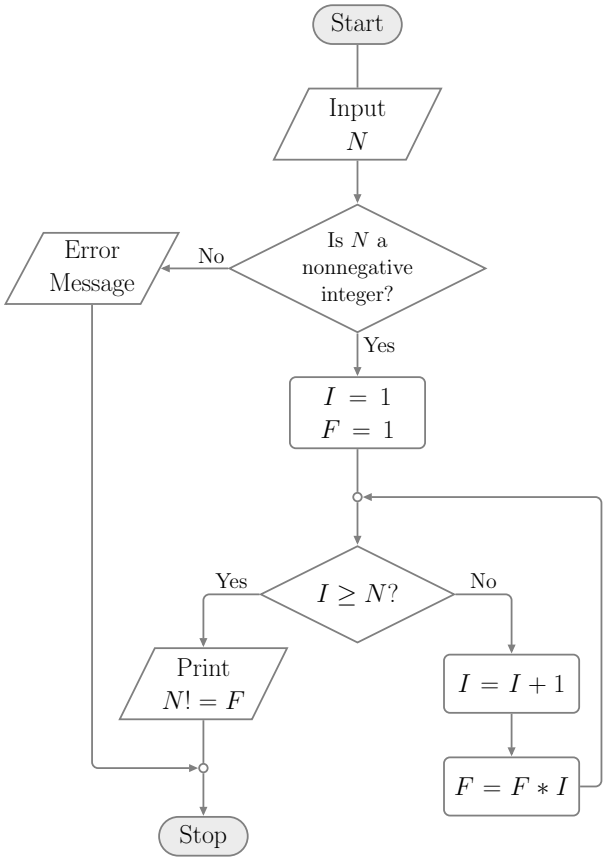


Figure 1: Case study: a flowchart to compute the factorial of a number.

Last time I talked about the *just-do-it-for-me* questions. Since a lot of them refer to Ti*k*Z, learning some tips and tricks could be useful.

I will show you some of them drawing the flowchart in Figure 1. I chose it because I often see questions about similar diagrams on TEX.SE.

Of course, as always in LATEX, there are many ways to skin a duck, er, to make such a diagram. Some of them are even more efficient and fun than the one I will show here, for example with a matrix or a chain (how many topics I do have for the next issues, quack!). But I would like to proceed one step at a time. I also will not be very rigorous, I hope the experts will forgive me.

Anyway, I thank all the users whose answers helped me to build the example. They are too many to list them all here, quack!

2 Origin and evolution of the *duck mania*

I would like to tell you a story which is both moving and funny; it concerns the origin of the duck mania. The protagonist is, of course, Paulo Cereda. I will

just report (more or less exactly) what he said in chat about it.

It happened that, one day, he was in a conference and saw a lone girl in the corner of the auditorium. He decided at once to talk to her (he never loses an opportunity to make new friends). But when he said "Hi!", she did not reply; she seemed not even to notice his presence. When he came near her, she looked at him, saying no words, and wrote in a piece of paper that she was hearing impaired. Of course, this was not a problem for Paulo, who promptly used his notebook to write sentences which she could read.

To impress her, he mentioned that he studied ASL (American Sign Language), even if he remembered almost nothing. When she asked him to try a sentence for her, he got stuck because the only words he could remember were: *I love you* and, of course, *duck*.

Since saying "I love you" to a girl you have known for just a few minutes is not very appropriate, he chose — in his own words — the second best sentence ever known to mankind: "I love ducks".

Eventually, after much gesticulation, the girl, obviously, started laughing a lot!

This good memory is the reason why Paulo began spreading the duck mania all over the world.

One way to infect other people was to offer a hand puppet duck as a prize in a TEX.SE (Meta) contest. I moved to Milan, to my friend Carla's, on that occasion (she won the contest).

Images or words related to ducks have been used in TEX.SE posts for years; some users have a duck as their avatar. The peak of the infection was reached with the creation of `tikzducks`,[1] and since the package is growing bigger and bigger, the duck joke will last for many years to come.

3 Quack Guide No. 2 The Morse code of Ti*k*Z

At first sight, Ti*k*Z may scare newbies due to its huge package documentation [1], but its usage is not so difficult as it may seem.

Its logic is simple: like the Morse code uses *dots* and *dashes* to translate any text, Ti*k*Z uses *nodes* and *paths* to draw any picture!

If you look at Figure 1, you will see some geometric shapes, connected by lines (in this case arrows): the former are nodes, the latter are paths. Are you looking forward to learning how to draw them? Just load the `tikz` package, add a `tikzpicture` environment to your document, and start!

[1] `https://ctan.org/pkg/tikzducks`.

3.1 Nodes

The syntax of the node command is more or less:

`\node[`⟨*options*⟩`] (`⟨*name*⟩`) at (`⟨*coord*⟩`) {`⟨*text*⟩`};`

Only `{`⟨*text*⟩`}`, i.e., the text within the node, is mandatory, although it can be empty: `{}`.

⟨*name*⟩ is the identifier by which the node will be referenced in your `tikzpicture`; it can also be set with the option `name=`⟨*name*⟩.

The coordinates where it will be located are (⟨*coord*⟩), they can be Cartesian, polar or spherical; the default is `(0,0)`.

As for ⟨*options*⟩, you can play around setting dimensions, aspect, positioning, labels, you name it. Of course, I cannot list all of them in these few pages, I will only highlight the ones who surprised me when I first met them, quack!

Figure 2 shows the options for setting the node dimensions. All of these are followed by the actual desired value, e.g., `inner sep=`⟨*dimension*⟩ and they are not mandatory; if not explicitly set, they assume a default value.

For instance, the default value for `text width` is the natural width of your node text; let us call the latter w, for convenience. If your node does not have an explicit `text width`, it is set to w, as in the first node of the following example. If a `text width` less than w is indicated, your text will be broken onto more than one line, as in the second node. If it is greater than w, the remaining space will be filled with spaces, as in the third node.

```
\begin{tikzpicture}[every node/.style={draw}]
  \node {We love ducks};
  \node[text width=4em] at (3,0)
    {We love ducks};
  \node[text width=10em] at (1.5,-1)
    {We love ducks};
\end{tikzpicture}
```

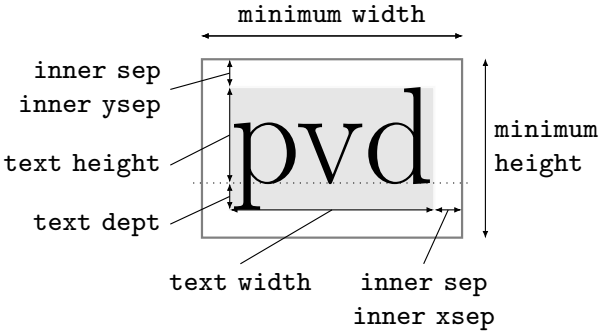
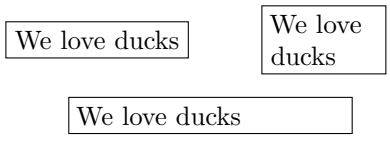


Figure 2: Node dimensions. The dotted line is the baseline; the text has a gray background to better highlight its dimensions. For the border style see Figure 5. *(The first one who guesses what* pvd *means wins a rubber duck.)*



The first strangeness you might notice in Figure 2 is that there are `minimum width/height` but *not* the corresponding maximum. Indeed, if you consider a node border with no thickness, the node width is the sum of the `text width` and the double of the `inner xsep`, whereas the node height is the sum of `text height`, `text depth` and the double of the `inner ysep`. Hence, for instance, you will usually act on the `text width` to set a maximum width.

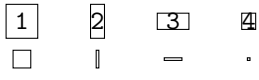
The `text height` is the piece of text above the baseline; the `text depth` is the one below.

The `inner sep` is the gap between the text and the node border. You can set the horizontal/vertical value separately with `inner xsep`/`inner ysep`.

If the border has a thickness, half of its thickness will be inside the shape and half outside, so to compute precisely the total node width/height you should also add the line width of the border.

Have you got a headache yet? Don't worry, quack! Let me show you an example:

```
\begin{tikzpicture}[
  every node/.style={draw, font=\ttfamily}
  ]
  \node {1};
  \node[inner xsep=0em] at (1,0) {2};
  \node[inner ysep=0em] at (2,0) {3};
  \node[inner sep=0em] at (3,0) {4};
  \node at (0,-.5) {};
  \node[inner xsep=0pt] at (1,-.5) {};
  \node[inner ysep=0pt] at (2,-.5) {};
  \node[inner sep=0pt] at (3,-.5) {};
\end{tikzpicture}
```



The nodes in the first column have the standard inner sep dimension (which is `.3333em`), with and without text. As you can see, even if there is no text, the node has a width and a height, due to the inner sep. In the second and third columns, there are nodes respectively with no horizontal and no vertical gaps between the text and the border, whereas, in the last column, there are no gaps at all. To have a point with no dimensions, you have to nullify also the `inner sep`, as in the last node (if you only need an actual geometric point, you can use `\coordinate`, but I will not talk about it this time).

Did you notice the options of the `tikzpicture` environment? `draw` means that you want the node borders visible, and you can set the font used for the node text with `font=`⟨*font commands*⟩.

Imagine that you have a picture with a lot of nodes — writing these options for every node could be boring! But LaTeX is fun; it is made to avoid code repetition, quack! The Ti*k*Z way to do this is to create a `style`. You can make the style valid for all the nodes, as in the previous example, or only for some of them, giving your own name to the style. In our case study, for instance, we will create a style for the terminal blocks, one for the instructions, another for the tests, and so on.

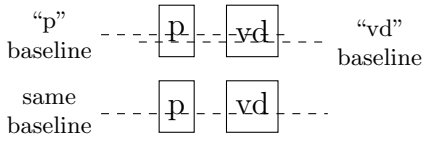
If you specify them, like in the previous example, as options of your `tikzpicture` environment, you can use them only locally. To make them valid for all the pictures of our document, you can use `\tikzset`, and write them in our preamble or anywhere before using them:

`\tikzset{`⟨*style name*⟩`/.style={`⟨*options*⟩`},...}`

Another example is a handy application of `text height` and `text depth`: alignment of texts in different nodes.

Look at this code snippet and its output (for now, do not worry about the `\draw` commands, I will explain them in Section 3.2):

```
\begin{tikzpicture}[
  mylabel/.style={font=\small, align=center},
  mynode/.style={draw, font=\large,
                 minimum height=4.5ex},
  mynodeok/.style={draw, font=\large,
                   text height=1.75ex,
                   text depth=.5ex,
                   minimum height=4.5ex}]
  \node[mynode] (p) {p};
  \node[mynode] (vd) at (1,0) {vd};
  \draw[dashed] (p.base) +(-1,0)
    node[mylabel, left] {``p'' \\ baseline}
    -- +(1.5,0);
  \draw[dashed] (vd.base) +(-1.5,0) -- +(1,0)
   node[mylabel, right] {``vd'' \\ baseline};
  \node[mynodeok] (pok) at (0,-1) {p};
  \node[mynodeok] at (1,-1) {vd};
  \draw[dashed] (pok.base) +(-1,0)
   node[mylabel, left] {same \\ baseline}
   -- +(2,0);
\end{tikzpicture}
```



It is evident that the nodes of the first row have different baselines, but adding the appropriate text height and depth, you get two nodes with a perfectly aligned text (second row).

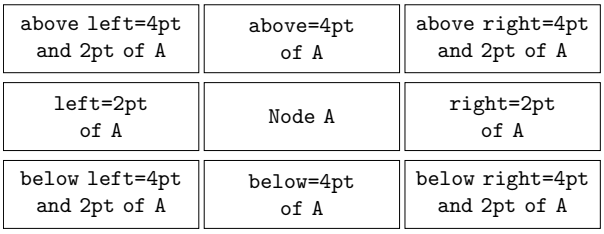


Figure 3: Node locating with Ti*k*Z library `positioning`. See Section 17.5 of [1].

The option `align=`⟨*alignment option*⟩, which I used for the side descriptions, sets up the alignment for multi-line text inside a node.

So far I have explicitly set the coordinates to locate the nodes. With a complex picture, it could be not only dull, but you may also be obliged to recalculate the coordinates of many nodes for a small change to your image, even if their relative positions, with respect to other nodes, remain the same.

In such cases, the Ti*k*Z library `positioning` could be your friend!

What is a Ti*k*Z library? You know that Ti*k*Z is a huge package, but usually you do not need all its possible features; a Ti*k*Z library allows you to load some specific additional ones. Just add

`\usetikzlibrary{`⟨*list of libraries*⟩`}`

after loading Ti*k*Z to use them.

Figure 3 shows some options you can use with `positioning`. The locating options (`above`/`below`, `right`/`left` and their combinations) are followed by a ⟨*shifting part*⟩ and an ⟨*of-part*⟩, and they are both optional. Please note that the equal sign must be located before the ⟨*of-part*⟩, even if the ⟨*shifting part*⟩ is not present.

In the ⟨*shifting part*⟩, you can indicate, for instance, a ⟨*dimension*⟩, which represents the distance between the borders of the nodes you would like to set. For the options like `above left`, you can also differentiate between the vertical and horizontal distances, writing ⟨*vdimension*⟩ `and` ⟨*hdimension*⟩. If they are the same for all your nodes, you could add a single option to your environment:

`node distance=`⟨*shifting part*⟩

In the ⟨*of-part*⟩, you can tell Ti*k*Z with respect to which node or coordinate your node should be placed.

To tell the truth, you could use `above` & Co. also without any library, and you can also use anchors (see below) to locate nodes. However, I still advise using `positioning` because it is simpler and has more features.

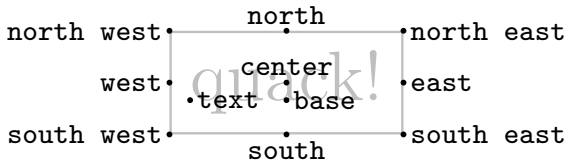
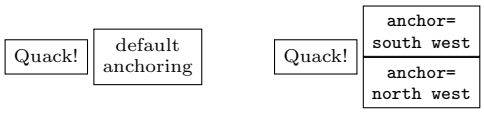


Figure 4: The main anchors of a rectangular node. For further details and other shapes see Section 67 of [1].

Figure 4 shows the main anchors of a rectangular shape. Other shapes may have other, possibly non-intuitive, anchors. Anchors could be used for node positioning (see the following example). In our case study, I will use one also as a starting point of a path. Indeed, anchors are genuine coordinates; you can refer to them with ⟨*node name*⟩`.`⟨*anchor*⟩.

```
\usetikzlibrary{positioning}
...
\begin{tikzpicture}[node distance=2pt,
  every node/.style={draw,
    align=center, font=\scriptsize}]
  \node (a) {Quack!};
  \node[right=of a] {default\\ anchoring};
  \node[right=7em of a] (b) {Quack!};
  \node[right=of b, font=\scriptsize\ttfamily,
    anchor=north west] {anchor=\\ north west};
  \node[right=of b, font=\scriptsize\ttfamily,
    anchor=south west] {anchor=\\ south west};
\end{tikzpicture}
```



With `positioning`, the default anchor for a node positioned to the right of another one is `west` (see the left side of the above picture), but you can change this behavior, setting an `anchor` option explicitly (as on the right side).

Ti*k*Z offers countless node shapes; `rectangular` is the default one. To draw our flowchart you also need a `circle`, which does not require any additional library, a `rounded rectangle`, for which you need `shapes.misc`, and, lastly, a `diamond` and a `trapezium` of `shapes.geometric`.

Some shapes may have additional options. For example, in our case study, we will modify the standard `trapezium` side angles; for the `diamond`, we will change the ratio between its width and height with `aspect=`⟨*number*⟩ (if the option is not present, it is set to 1).

Let us see some options to color our nodes: `text=`⟨*color*⟩ colors the text, `draw=`⟨*color*⟩ colors the borders, simply ⟨*color*⟩ colors both, whereas for the background there is `fill=`⟨*color*⟩.

In the following example I am using `lightgray` for editorial reasons, but, of course, you can use any color you prefer.

```
\begin{tikzpicture}[every node/.style={draw,
    font=\scriptsize}]
 \node[text=lightgray] (a) {Quack!};
 \node[right=of a, draw=lightgray](b){Quack!};
 \node[right=of b, lightgray] (c) {Quack!};
 \node[right=of c, fill=lightgray] {Quack!};
\end{tikzpicture}
```

3.2 Paths

Having created and located our nodes, let us learn the command to link them:

`\path[`⟨*option*⟩`]` ⟨*path specification*⟩`;`

Since almost all paths are drawn, there is also the abbreviation `\draw` for `\path[draw]`.

For ⟨*option*⟩, again, you can put whatever you like for changing the aspect of the line. There are also a lot of path specifications. Here I will show only the ones used in our case study.

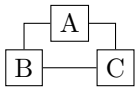
The simplest one is:

`\draw (`⟨*starting point*⟩`) -- (`⟨*target*⟩`);`

which produces a straight line; the ⟨*starting point*⟩ and the ⟨*target*⟩ could be nodes or coordinates, and it is also possible to add other points to the path.

If you use `|-` or `-|`, instead of a straight line you will have a line with a 90° angle, respectively starting vertically and going on horizontally, or vice-versa; see the following example:

```
\begin{tikzpicture}[every node/.style=draw,
  node distance=4pt]
  \node (a) {A};
  \node[below left=of a] (b) {B};
  \node[below right=of a] (c) {C};
  \draw (a) -| (b) -- (c) |- (a);
\end{tikzpicture}
```



The line thickness and its pattern can be customized extensively; you will find some examples in Figure 5. If your desired thickness is not among the predefined ones, you can set it to any value you like with `line width=`⟨*dimension*⟩.

A useful feature is the possibility of indicating a coordinate of the path relative to another point, with

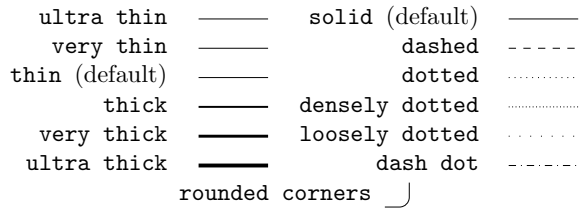


Figure 5: Examples of path thicknesses and patterns. The same options are valid for node borders. See Section 15.3 of [1] for more details.

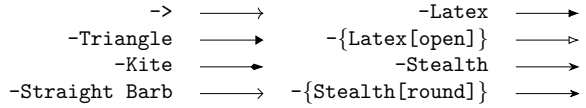


Figure 6: Arrow tips, see Section 16.5 of [1] for a complete list and options.

`+(`⟨*shift*⟩`)` and `++(`⟨*shift*⟩`)`. The difference between the two notations is that `++` updates the current point while `+` does not. The current point is the one from which the ⟨*shift*⟩ will be applied; of course, it could be either positive or negative, and any kind of dimension or coordinates can be used. Let us look at an example:

```
\begin{tikzpicture}
  \draw (0,0) -- ++(1,-2) -- +(2,1);
  \draw[dashed] (0,0) -- +(1,-2) -- +(2,1);
\end{tikzpicture}
```

The first segments coincide because the second vertex is `(0,0)+(1,-2)=(1,-2)` for both the solid and the dashed paths; whereas the second segments differ: indeed, the last vertex is `(1,-2)+(2,1)=(3,-1)` for the solid path whereas `(0,0)+(2,1)=(2,1)` for the dashed one.

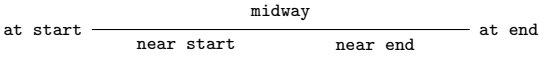
In our case study, however, there are not simple lines but arrows. The Ti*k*Z library `arrow.meta` provides many different arrow tips, which are then further customizable.

In our flowchart, I used a triangular tip with a smaller width: `-{Triangle[width=5pt]}`. You will find some other examples in Figure 6. To create an arrow path it is enough to put the kind of arrow tip you like in one of these ways: `-`⟨*arrow tip*⟩, ⟨*arrow tip*⟩`-`, or ⟨*arrow tip*⟩`-`⟨*arrow tip*⟩, depending on if you need the tip at the beginning of your path, at the end of it or both.

Eventually, it may be useful to put some nodes along the path; see, for instance, the "Yes" and "No" exits of the tests of our flowchart. It can be done easily by putting a `node` (without the backslash

because it is not a macro but a path option) in an appropriate position. Note also where the semicolon is positioned in the following example:

```
\begin{tikzpicture}[
  every node/.style={font=\scriptsize\ttfamily}
  ]
  \draw (0,0) -- +(5.5,0)
    node[at start, left] {at start}
    node[near start, below] {near start}
    node[midway, above] {midway}
    node[near end, below] {near end}
    node[at end, right] {at end};
\end{tikzpicture}
```



3.3 Let us put them together

Now you have all the tools needed to understand the complete code of our case study.

```
\documentclass[tikz]{standalone}
\usetikzlibrary{positioning,
  shapes.geometric, shapes.misc, arrows.meta}
\begin{document}
\begin{tikzpicture}[
  every path/.style={gray, very thick,
rounded corners,-{Triangle[width=5pt]}},
  basenode/.style={draw, sharp corners,
text=black},
  terminator/.style={basenode,font=\LARGE,
rounded rectangle,minimum height=6ex,
text width=5em,text height=2.25ex,
    text depth=.25ex,
    fill=lightgray!30,align=center,},
  inout/.style={basenode,font=\LARGE,
    text width=5.8em,minimum height=11ex,
    align=center,trapezium,trapezium stretches,
trapezium left angle=60,
trapezium right angle=120,},
  block/.style={basenode,font=\LARGE,
    text width=9em,minimum height=9ex,
rounded corners,inner sep=0pt,
    align=center,},
  decision/.style={basenode,diamond,
    align=flush center,aspect=2,font=\LARGE,
    minimum height=15ex,minimum width=30ex,
inner sep=0pt,},
  joining/.style={basenode,circle,
    inner sep=2pt,},
  yesno/.style={font=\Large,near start,black},
  ]
% nodes
  \node[terminator](start) {Start};
  \node[inout, below=of start](input)
    {Input\\ $N$};
  \node[decision, below=of input,
    font=\Large](dqtest)
    {Is $N$ a\\ nonnegative \\ integer?};
  \node[inout, left=4.5em of dqtest](error)
    {Error Message};
  \node[block, below=of dqtest,
    minimum height=11ex](setvar)
    {$I = 1$\\$F = 1$};
  \node[joining, below=of setvar](join1){};
  \node[decision, below=of join1](looptest)
    {$I \ge N$?};
  \node[inout,
    below left=10ex and 7em of looptest,
    anchor=center](output){Print\\ $N! = F$};
  \node[block,
    below right= 10ex and 7em of looptest,
    anchor=center](increase){$I = I + 1$};
  \node[joining, below=of output](join2){};
  \node[block, below=of increase](multiply)
    {$F = F \ast I$};
  \node[terminator, below=of join2](stop)
    {Stop};
% paths
  \draw (start) -- (input) -- (dqtest);
  \draw (dqtest) -- node[yesno, right] {Yes}
    (setvar);
  \draw (dqtest) -- node[yesno, above] {No}
    (error);
  \draw (setvar) -- (join1) -- (looptest);
  \draw (looptest) -| node[yesno, above] {Yes}
    (output);
  \draw (looptest) -| node[yesno, above] {No}
    (increase);
  \draw (output) -- (join2) -- (stop);
  \draw (error) |- (join2);
  \draw (increase) -- (multiply);
  \draw (multiply.east) -- +(.5,0) |- (join1);
\end{tikzpicture}
\end{document}
```

4 Conclusions

I hope you liked my explanation, and if you have trouble in using Ti*k*Z, remember:

> ***A duck makes you laugh!***

References

[1] Till Tantau. The Ti*k*Z and PGF packages. http://mirrors.ctan.org/graphics/pgf/base/doc/pgfmanual.pdf. Package page: https://ctan.org/pkg/pgf.

◇ Herr Professor Paulinho van Duck
Quack University Campus
Sempione Park Pond, Milano
Italy
paulinho dot vanduck (at) gmail dot com

Solutions for the CTAN quiz (on p. 18)

1. https://ctan.org/topics/cloud
2. https://ctan.org/lion
3. https://ctan.org/credits
4. https://ctan.org/help/supported-browsers
5. https://ctan.org/mirrors
6. https://ctan.org/mirrors/register
7. https://ctan.org/pkg
8. https://ctan.org/author
9. https://ctan.org/ctan-ann
10. https://ctan.org/search?ext=new
11. https://ctan.org/upload
12. https://ctan.org/help/submit
13. https://ctan.org/user/settings
14. https://ctan.org/author/knuth
15. https://ctan.org/help/json
16. https://ctan.org/tex-archive
17. https://ctan.org/home
18. https://ctan.org/guestbook
19. https://ctan.org/lugs
20. https://ctan.org/incoming

Comic by John Atkinson (http://wronghands1.com).

From Lua 5.2 to 5.3

Hans Hagen

When we started with LuaTEX we used Lua 5.1 and moved to 5.2 when that became available. We didn't run into issues then because there were no fundamental changes that could not be dealt with. However, when Lua 5.3 was announced in 2015 we were not sure if we should make the move. The main reason was that we'd chosen Lua because of its clean design which meant that we had only one number type: double. In 5.3 on the other hand, deep down a number can be either an integer or a floating point quantity.

Internally TEX is mostly (up to) 32-bit integers and when we go from Lua to TEX we round numbers. Nonetheless one can expect some benefits in using integers. Performance-wise we didn't expect much, and memory consumption would be the same too. So, the main question then was: can we get the same output and not run into trouble due to possible differences in serializing numbers; after all TEX is about stability. The serialization aspect is for instance important when we compare quantities and/or use numbers in hashes.

Apart from this change in number model, which comes with a few extra helpers, another extension in 5.3 was that bit-wise operations are now part of the language. The lpeg library is still not part of stock Lua. There is some minimal UTF8 support, but less than we provide in LuaTEX already. So, looking at these changes, we were not in a hurry to update. Also, it made sense to wait till this important number-related change was stable.

But, a few years later, we still had it on our agenda to test, and after the ConTEXt 2017 meeting we decided to give it a try; here are some observations. A quick test was just dropping in the new Lua code and seeing if we could make a ConTEXt format. Indeed that was no big deal but a test run failed because at some point a (for instance) `1` became a `1.0`. It turned out that serializing has some side effects. And with some ad hoc prints for tracing (in the LuaTEX source) I could figure out what went on. How numbers are seen can (to some extent) be deduced from the `string.format` function, which is in Lua a combination of parsing, splitting and concatenation combined with piping to the C code `sprintf` function.[1]

[1] Actually, at some point I decided to write my own formatter on top of `format` and I ended up with splitting as well. It's only now that I realize why this is working out so well (in terms of performance): simple format (single items) are passed more or less directly to `sprintf` and as Lua itself is

```
local a =  2   * (1/2) print(string.format("%s",  a),math.type(x))
local b =  2   * (1/2) print(string.format("%d",  b),math.type(x))
local c =  2           print(string.format("%d",  c),math.type(x))
local d = -2           print(string.format("%d",  d),math.type(x))
local e =  2   * (1/2) print(string.format("%i",  e),math.type(x))
local f =  2.1         print(string.format("%.0f",f),math.type(x))
local g =  2.0         print(string.format("%.0f",g),math.type(x))
local h =  2.1         print(string.format("%G",  h),math.type(x))
local i =  2.0         print(string.format("%G",  i),math.type(x))
local j =  2           print(string.format("%.0f",j),math.type(x))
local k = -2           print(string.format("%.0f",k),math.type(x))
```

number	fmt	out	type
2 * (1/2)	s	1.0	float
2 * (1/2)	d	1	float
2	d	2	integer
−2	d	2	integer
2 * (1/2)	i	1	float
2.1	.0f	2	float
2.0	.0f	2	float
2.1	G	2.1	float
2.0	G	2	float
2	.0f	2	integer
−2	.0f	2	integer

Figure 1: Various number representation in Lua 5.3: code at left, summary and output at right.

Figure 1 gives many examples, demonstrating that we have to be careful when we need these numbers represented as strings. In ConTEXt the number of places where we had to check for that was not that large; in fact, only some hashing related to font sizes had to be done using explicit rounding.

Another surprising side effect is the following. Instead of:

```
local n = 2^6
```

we now need to use:

```
local n = 0x40
```

or just:

```
local n = 64
```

because we don't want this to be serialized to `64.0` which is due to the fact that a power results in a float. One can wonder if this makes sense when we apply it to an integer.

At any rate, once we could process a file, two documents were chosen for a performance test. Some experiments with loops and casts had demonstrated that we could expect a small performance hit and indeed, this was the case. Processing the LuaTEX manual takes 10.7 seconds with 5.2 on my 5-year-old laptop and 11.6 seconds with 5.3. If we consider that ConTEXt spends 50% of its time in Lua, then we see a 20% performance penalty. Processing the Metafun manual (which has lots of MetaPost images) went from less than 20 seconds (LuaJITTEX does it in 16 seconds) up to more than 27 seconds. So there we lose more than 50% on the Lua end. When we observed these kinds of differences, Luigi and I immediately got into debugging mode, partly out of curiosity, but also because consistent performance is important to us.

fast, due to some caching, the overhead is small compared to the built-in splitter method. And the ConTEXt formatter has many more options and is extensible.

Because these numbers made no sense, we traced different sub-mechanisms and eventually it became clear that the reason for the speed penalty was that the core `string.format` function was behaving quite badly in the `mingw` cross-compiled binary, as seen by this test:

```
local t = os.clock()
for i=1,1000*1000 do
 -- local a = string.format("%.3f",1.23)
 -- local b = string.format("%i",123)
    local c = string.format("%s",123)
end
print(os.clock()-t)
```

	lua 5.3	lua 5.2	texlua 5.3	texlua 5.2
a	0.43	0.54	3.71 (0.47)	0.53
b	0.18	0.24	3.78 (0.17)	0.22
c	0.26	0.68	3.67 (0.29)	0.66

The 5.2 binaries perform the same but the 5.3 Lua binary greatly outperforms LuaTEX, and so we had to figure out why. After all, all this integer optimization could bring some gain! It took us a while to figure this out. The numbers in parentheses are the results after fixing this.

Because font internals are specified in integers one would expect a gain in running:

```
mtxrun --script font --reload force
```

and indeed that is the case. On my machine a scan results in 2561 registered fonts from 4906 read files and with 5.2 that takes 9.1 seconds while 5.3 needs a bit less: 8.6 seconds (with the bad format performance) and even less once that was fixed. For a test:

```
\setupbodyfont[modern]     \tf \bf \it \bs
\setupbodyfont[pagella]    \tf \bf \it \bs
\setupbodyfont[dejavu]     \tf \bf \it \bs
\setupbodyfont[termes]     \tf \bf \it \bs
\setupbodyfont[cambria]    \tf \bf \it \bs
\starttext \stoptext
```

This code needs 30% more runtime so the question is: how often do we call `string.format` there? A first run (when we wipe the font cache) needs some 715,000 calls while successive runs need 115,000 calls so that slow down definitely comes from the bad handling of `string.format`. When we drop in a Lua update or whatever other dependency we don't want this kind of impact. In fact, when one uses external libraries that are or can be compiled under the TEX Live infrastructure and the impact would be such, it's bad advertising, especially when one considers the occasional complaint about LuaTEX being slower than other engines.

The good news is that eventually Luigi was able to nail down this issue and we got a binary that performed well. It looks like Lua 5.3.4 (cross)compiles badly with GCC 5.3.0 and 6.3.0.

So in the end caching the fonts takes:

	caching	running
5.2 stock	8.3	1.2
5.3 bugged	12.6	2.1
5.3 fixed	6.3	1.0

So indeed it looks like 5.3 is able to speed up LuaTEX a bit, given that one integrates it in the right way! Using a recent compiler is needed too, although one can wonder when a bad case will show up again. One can also wonder why such a slow down can mostly go unnoticed, because for sure LuaTEX is not the only compiled program.

The next examples are some edge cases that show you need to be aware that 1) an integer has its limits, 2) that hexadecimal numbers are integers and 3) that Lua and LuaJIT can be different in details.

```
print(0xFFFFFFFFFFFFFFFF)
```

lua 5.2	1.844674407371e+019
luajit	1.844674407371e+19
lua 5.3	−1

```
print(0x7FFFFFFFFFFFFFFF)
```

lua 5.2	9.2233720368548e+018
luajit	9.2233720368548e+18
lua 5.3	9223372036854775807

So, to summarize the process. A quick test was relatively easy: move 5.3 into the code base, adapt a little bit of internals (there were some LuaTEX interfacing bits where explicit rounding was needed), run tests and eventually fix some issues related to the Makefile (compatibility) and C code obscurities (the slow `sprintf`). Adapting ConTEXt was also not much work, and the test suite uncovered some nasty side effects. For instance, the valid 5.2 solution:

```
local s = string.format("02X",u/1024)
local s = string.char        (u/1024)
```

now has to become (both 5.2 and 5.3):

```
local s = string.format("02X",math.floor(u/1024))
local s = string.char        (math.floor(u/1024))
```

or (both 5.2 and (emulated or real) 5.3):

```
local s = string.format("02X",bit32.rshift(u,10))
local s = string.char        (bit32.rshift(u,10))
```

or (only 5.3):

```
local s = string.format("02X",u >> 10))
local s = string.char        (u >> 10)
```

or (only 5.3):

```
local s = string.format("02X",u//1024)
local s = string.char        (u//1024)
```

A conditional section like:

```
if LUAVERSION >= 5.3 then
  local s = string.format("02X",u >> 10))
  local s = string.char        (u >> 10)
else
  local s = string.format("02X",
                           bit32.rshift(u,10))
  local s = string.char  (bit32.rshift(u,10))
end
```

will fail because (of course) the 5.2 parser doesn't like that. In ConTEXt we have some experimental solutions for that but that is beyond this summary.

In the process a few UTF helpers were added to the string library so that we have a common set for LuaJIT and Lua (the `utf8` library that was added to 5.3 is not that important for LuaTEX). For now we keep the `bit32` library on board. Of course we'll not mention all the details here.

When we consider a gain in speed of 5-10% with 5.3 that also means that the gain of LuaJITTEX compared to 5.2 becomes less. For instance in font processing both engines now perform closer to the same.

As I write this, we've just entered 2018 and after a few months of testing LuaTEX with Lua 5.3 we're confident that we can move the code to the experimental branch. This means that we will use this version in the ConTEXt distribution and likely will ship this version as 1.10 in 2019, where it becomes the default. The 2018 version of TEX Live will have 1.07 with Lua 5.2 while intermediate versions of the Lua 5.3 binary will end up on the ConTEXt garden, probably with number 1.08 and 1.09 (who knows what else we will add or change in the meantime).

◇ Hans Hagen
Pragma ADE
`http://pragma-ade.com`

TEXing in Emacs

Marcin Borkowski

Abstract

In this paper I describe how I use GNU Emacs to work with LaTeX. It is not a comprehensive survey of what can be done, but rather a subjective story about my personal usage.

In 2017, I gave a presentation [1] during the joint GUST/TUG conference at Bachotek. I talked about my experiences typesetting a journal (*Wiadomości Matematyczne*, a journal of the Polish Mathematical Society), and how I utilized LaTeX and GNU Emacs in my workflow. After submitting my paper to the proceedings issue of *TUGboat*, Karl Berry asked me whether I'd like to prepare a paper about using Emacs with LaTeX.

Well, I jumped at the proposal. I am a great fan of Emacs, and I've been using it for nearly two decades now. So, here is my Emacs/TeX story.

I decided to divide this tale in four parts. The zeroth one is a very brief explanation of how I got where I am with respect to Emacs. The first one is a very short introduction to the main concepts and terminology of Emacs. Then, I talk about various Emacs packages I use in my day-to-day (La)TeX work. Finally, for the brave souls who would like to go deeper into the Emacs rabbit hole, I present a few example snippets I wrote to make Emacs suit my personal needs.

0 The beginnings

GNU Emacs is an ancient piece of software, started by the famous Richard M. Stallman, and used right through today. (Interestingly, one of the reasons Stallman started the GNU project, which GNU Emacs soon became part of, was ethical rather than technical. He has a very distinct set of moral beliefs, one of which is that everyone should have certain freedoms with respect to the software they use, and sticks to them *without compromise*. Disclaimer: while I share some, but not all of his views, I still admire his perseverance and his strong conviction about the objectivity of moral principles — even if he gets some of these wrong.)

When I started using Emacs (note: I will not say "GNU Emacs" each time; nowadays, there are essentially no other Emacsen, since the last "competitor", XEmacs, seems to have died a few years ago), I needed just a text editor for TeX (I was using plain TeX at the time). When I switched from DOS and MS Windows 3.11 to a GNU/Linux system, I heard that there are two editors, and had to choose one of them. I used a simple criterion: Emacs had a nice tutorial, and Vim apparently did not (at that time).

I wince at the very thought I might have chosen wrong!

And so it went. I started with reading the manual [8]. As a student, I had *a lot* of free time on my hands, so I basically read most of it. (I still recommend that to people who want to use Emacs seriously.) I noticed that Emacs had a nice TeX mode built-in, but also remembered from one of the BachoTeXs that other people had put together something called AUCTeX, which was a TeX-mode on steroids.

In the previous paragraph, I mentioned *modes*. In order to understand what an Emacs mode is, let me explain what this whole Emacs thing is about.

1 Basics of Emacs

It is actually easy to understand Emacs. (Do not confuse "understand" with "master", by the way.) There are many ways of explaining the phenomenon of a piece of software which has existed for about four decades and is still relevant today. A popular view (and a subject of jokes) is that Emacs is an operating system disguised as an editor. This is surprisingly close to the truth, but this metaphor does not help with talking about how Emacs interacts with (La)TeX. Another known aphorism is that Emacs is not a text editor, but a DIY kit for creating *your* own editor, suited to your needs. This is also true, and I will come back to it later. However, I think that in order to explain Emacs, I should describe the core concepts which make it what it is. For those, I choose three basic Emacs notions of *buffers*, *commands* and *keybindings*, and a fourth one which, well, *binds* them together: modes.

Just as in Unix "everything is a file", in Emacs "everything is a buffer". Interestingly, an Emacs buffer is an entity which is quite close to a Unix file. It is identified by name, and it consists of characters. (There's more to it than that, but let's keep things simple.) If you visit a file in Emacs (this is what most other editors call "opening" a file), a buffer with a name corresponding to the name of the file is created and filled with the characters read from that file. From now on, Emacs does not care about a physical file on the disk (or somewhere else — Emacs can also open files located "in the cloud", i.e., on other people's computers), at least not until we want to save the buffer to the file again.

In most (La)TeX editors (and most other applications, for that matter), we have things such as dialog windows, non-editable text areas with the compilation log, file-selecting widgets, etc. Not in Emacs:

here, all these are buffers (some of them read-only, of course). Here is an example: if you press `C-x d RET` (in Emacs parlance, this means pressing `Control-X`, then `D` and then `Enter`), you will see a Dired *buffer*, which contains a listing of files in the current directory, much like the output of the Unix `ls` command. It is not normally editable, but you can use various keybindings to move *point* (i.e., the cursor), and perform various actions on the listed files, like visiting them (`f` or `RET`), copying them somewhere else (`C`, i.e., `Shift-C`), diffing them with other files (`=`), etc.

Let us now talk about *commands*. They are pieces of code (usually in Emacs Lisp, or Elisp, the language the majority of Emacs is written in) which perform various tasks you would expect from a text editor (and more). For instance, there is a command called `find-file`, which asks the user for a file name and visits it in an Emacs buffer. Another, called `save-buffer`, saves the contents of the current buffer to a file. Yet another, called `pong`, is an implementation of the old arcade classic.

It is important to understand that basically *any* action you perform in Emacs is a result of running some command. If you press some key (or key combination), Emacs checks the *binding* of that key, which says what command is *bound* to that key, and runs that command. This works even for very basic things, like the command `forward-char` (usually bound to `C-f` and `<right>`, that is, the right-arrow key), or keys with printable characters, which are usually bound to `self-insert-command`. A command need not be bound to a key — you can also call it by its name (for instance, the `pong` command is usually run by `M-x pong`, which means pressing `Alt-X`, then typing `pong` — the name of the command — then pressing `Enter`).

The last piece of this puzzle are *modes*, which tell Emacs the bindings of keys to commands in any particular buffer. (Each buffer has its associated mode, so switching buffers changes keybindings dynamically. Of course, there are also *global* bindings, which work everywhere, like `C-x C-c`, which exits Emacs.) For instance, in TEX mode, `C-c C-c` is bound to a command which does the "next logical thing", like compiling the file, running BIBTEX or launching a PDF viewer. The same key combination, `C-c C-c`, sends a message in a mode designed to write emails.

Finally, any introduction like this one would be incomplete without mentioning the self-documenting nature of Emacs. If you are in a buffer with some Emacs Lisp code, you can press `C-h f` when the point is on a function name, or `C-h v` when it is on a variable name, and immediately see the docstring for that function or variable, or even jump to the place in the source where it is defined. This is extremely useful, and there are specialized Emacs packages which streamline this even further (like showing the docstring in a tool-tip-like fashion, or finding all places where a function is called). Also, when you are in the middle of a function expression, Emacs shows the list of parameters of the current function at the bottom, with the parameter the point is on set in boldface. Finally, there is the whole set of *apropos* commands, which show all functions or variables matching the given regex — and "matching" can mean matching the name, the docstring, or even the value in case of variables. At any time, you can press `C-h C-h` to see the extensive list of Emacs help subsystems.

2 Emacs packages and features for TEXnicians

In this section, I am going to describe a few well-known (at least in the Emacs world) packages and features Emacs has to offer for people dealing with (LA)TEX. Note that this is not a comprehensive list — these are just the ones I happen to use.

2.1 AUCTEX

Although Emacs has a TEX mode built-in (and there are people using it), it is rather bare-bones. Happily, there is AUCTEX, a well-known package which is used by the majority of Emacsers who need to do stuff in TEX and friends.

Of course, AUCTEX has all the things you would expect, like auto-completing macro and environment names (it even knows the syntax of many LATEX commands and asks for the parameters, and parses `\newcommand`s to insert the proper number of braces after user-defined macros), commands to compile the file we are editing or syntax coloring (called "font-lock" in the Emacs world). It also has, however, some features which I believe are unique to it (although I admit that I have not used other editors extensively). AUCTEX has a very good manual (as do many Emacs packages — documenting things for users is emphasized a lot in the Emacs world), so let me just mention my personal favorite feature. You can select a portion of text and press `C-c C-r`. AUCTEX then writes a temporary file consisting of the preamble, the selection and `\end{document}`, compiles it and prepares the next viewing command to show this PDF instead of the whole file. Imagine a Beamer presentation with more than a thousand pages and a lot of drawings, which compiles for a few minutes in its entirety (true story!), and you can see what a life-saver this can be.

There is a lot more to AUCTEX than this. Let me mention LATEX-Math mode, which makes the key combinations starting with a backtick (`) insert many mathematical commands, so that you can get `\alpha` by pressing `a or `\emptyset` by pressing `0. Another feature of Emacs, which AUCTEX utilizes, is `prettify-symbols-mode`, which displays things like `\alpha` or `\int` using Unicode characters, which renders the source of math formulas much easier to read. (While at that, let me mention that despite its age, Emacs has excellent Unicode support, but can also handle some other, nowadays less popular encodings.) Yet another thing AUCTEX leverages is Emacs' `compilation-mode`, which parses the log buffer and allows you to, for instance, jump to the next or previous error. Basically, you have all you would expect from a good TEX editor. (One caveat is that support for plain TEX and ConTEXt is rather rudimentary.)

There is also a package called RefTEX, which is part of AUCTEX, whose goal is to help with all the references. One of its coolest features is showing the bibliographic info about a reference when the point is on the `\cite` macro. Unfortunately, RefTEX does not work with AMSrefs, and hence I do not use it personally (yet).

2.2 pdf-tools

There has been a PDF viewer in Emacs for a long time, but it never worked too well: under the hood, it just converted PDFs to bitmaps using Ghostscript and displayed them. Not very impressive, and very slow. Recently, however, another PDF viewer for Emacs was written, called pdf-tools. It is a wonderful piece of software, and even though it has its quirks (it does not have "spread" or "continuous" modes, for instance), it almost completely replaced Evince for me. It supports the SyncTEX extensions, thus allowing for jumping between the source and the PDF with very good accuracy, and allows for incremental search in a PDF (also with regexen). It also supports an Emacs concept called *occur*, which asks the user for a regex and displays a list of lines in the buffer matching that regex, allowing to jump immediately to any of these lines. You can also make Emacs watch for changes in the PDF and refresh it automatically, or issue a refresh when the compilation is finished. Perhaps the killer feature of pdf-tools, however, is its support for PDF annotations: it is possible to view all annotations without moving your hands from the keyboard.

This is also a good place to mention one of the many ways Emacs is flexible. In many places in the Emacs source code there are so-called *hooks*. They are variables, usually empty by default, which contain functions to be run at various moments. The pdf-tools package has a hook which contains functions run each time an annotation is shown. You can, for instance, add a function which checks whether the annotation is written in LATEX syntax, and if yes, call LATEX and then `dvipng` to display a picture of the typeset formula of something instead of the text of the annotation. In fact, a function doing exactly this is provided as an example, so I can actually format my PDF annotations in LATEX and pdf-tools just displays them correctly! There are about 150 hooks in stock Emacs, and many packages add their own — my Emacs has more than 700.

2.3 Other useful Emacs features

What makes Emacs an even better (LA)TEX editor are its features as a general text editor, applied to the particular case of (LA)TEX. I have already mentioned `compilation-mode` and `prettify-symbols-mode`; there is also a spell-checker (delegating its job to external tools, but the integration is seamless) and `auto-fill-mode`, which automatically wraps long lines using hard newlines (which is an abomination to many and a no-brainer for others). Emacs, however, is used by many people to edit texts in human languages (as opposed to computer programs), and has really good support for that task. For instance, while many editors have support for movement by words, Emacs has also movement by sentences. Another feature which is a real time-saver is the series of `transpose` commands: for instance, `transpose-chars` swaps the two characters on both sides of the point. There is a whole chapter in the Emacs manual describing commands for dealing with text in human languages.

Another Emacs tool which is used by many people is Yasnippet. It is a very useful tool for creating and inserting *snippets*, i.e., templates with placeholders for variable fragments. It is very easy to define one's own snippets. (Personally, I do not use Yasnippet with AUCTEX very often, since the latter can insert a lot of things for me, but I have snippets for, e.g., preambles of some documents.)

Last but not least, let me mention the Avy package. It solves the problem of quickly navigating to any place on the screen without using any pointing device (many Emacs users have an aversion to rodents and prefer using their keyboards as much as possible). The classical Emacs way has always been *isearch*, or *incremental search*: you can press `C-s` and start to type, and the point moves to the nearest occurrence of the typed character sequence while you

enter more and more characters (and Emacs highlights all occurrences thereof). Avy, which is based on ace-jump (a previous implementation of the same idea, which in turn was based on the *EasyMotion* Vim plugin), implements a simple but powerful concept. You can invoke the `avy-goto-char` command (many people bind it to some convenient key), then press a character key and all instances of this character on the screen become highlighted with a letter or a combination of letters. Pressing the letter (or a sequence of letters) moves the point to the respective location. It is an extremely fast and very convenient way of navigating, and also has many variants (like only selecting letters at the beginning of words or jumping to beginnings of lines).

2.4 Org-mode and LaTeX

In recent years, a new invention took the Emacs world by storm: Org-mode. Originally a note-taking mode on steroids, it quickly gained more and more features and is now a full-fledged application written on top of Emacs. (What saves it from the usual symptoms of *featuritis* is one of its design goals: advanced features should never get in the way if you do not want to use them.) It is difficult to describe Org-mode, since it combines a notebook, a literate programming environment (capable even of chaining pieces written in different languages!), a todo-list, a spreadsheet, a time-tracking tool and a few other things. What is interesting for TeX users is that Org-mode (which defines a markup syntax, quite similar to Markdown) has something called an *exporter*, which can save the document as a LaTeX article (or book), a Beamer presentation, an HTML page or even a LibreOffice document (and a few other, more obscure formats). Seasoned LaTeX users might scoff at such an idea and claim that one does not get the full power of LaTeX — and they would be right to some extent. However, for many people Org-mode syntax is much friendlier than LaTeX's, and most scientists do not use advanced TeX features anyway. (Also, you can embed arbitrary LaTeX stuff in an Org-mode document.) There is one place, however, where Org-mode is clearly superior to LaTeX: tables. Table editing in LaTeX is far from pleasant (although Emacs can help with that with automatic alignment of the table source on ampersand characters), and table sources are not very legible. On the other hand, see figure 1 for a table written in Org-mode and what Org-mode made with it when asked to export to LaTeX. Note that Org-mode has very good support for ASCII table editing — it takes care for making columns wide enough to accommodate the widest entry, for instance. Also, please note the last line, which has two

```
| Product | Price | Quantity | Amount |
|---------+-------+----------+--------|
| Bread   |  4.50 |        1 |   4.50 |
| Apples  |  2.40 |        4 |   9.60 |
| Tea     |  6.99 |        2 |  13.98 |
|---------+-------+----------+--------|
|         |       |    Total |  28.08 |
#+TBLFM: $4=$-1*$-2;f2::@5$4=vsum(@I..@II);f2

\begin{center}
\begin{tabular}{lrrr}
Product & Price & Quantity & Amount\\
\hline
Bread & 4.50 & 1 & 4.50\\
Apples & 2.40 & 4 & 9.60\\
Tea & 6.99 & 2 & 13.98\\
\hline
 &  & Total & 28.08\\
\end{tabular}
\end{center}
```

Figure 1: Org-mode table and the result of LaTeX exporting

formulæ for totaling. While an Org-mode spreadsheet does not feature automatic recalculation (it has to be triggered by issuing a special command), it is capable of performing quite advanced calculations — Org-mode utilizes Calc, a scientific calculator written in Elisp, for that.

3 Emacs customization

One of the most prominent features of Emacs is its flexibility. (In the first sentence of the Emacs manual it is called "the extensible, customizable, self-documenting real-time display editor".) There are literally thousands of options even in stock Emacs (without any packages loaded, Emacs has more than 2 000 variables; my Emacs has almost 15 000), and most packages add their own share. All these options can be set up using a nice and discoverable interface or manually, by editing an initialization file. For instance, AUCTeX by default asks the user whether to save the (La)TeX file before compilation. One can set, however, the `TeX-save-query` variable to nil (which is Elisp for "false"), and from then on the saving would be automatic.

Where Emacs really shines, though, is not in its *customizability*, but in its *extensibility*. Emacs has a small core written in C, but the rest is written in Elisp. In a properly configured Emacs, the source code for any command is a few keystrokes away, and you can modify its behavior within seconds. Of course, this requires knowledge of Emacs Lisp — but it is not a difficult language, and you can learn the basics within a few afternoons. There is an excellent

book [2], which is an introduction to Elisp for non-programmers. There is also the book [3], which is kind of a next step, although parts of it seem to be pretty outdated.

For the rest of this article I am thus going to talk about how you can mold Emacs to fit your needs. It is a selection of snippets of code which show the customizability and extensibility of Emacs.

I cannot resist to mention here that while writing this very paper I was advised against spending more time on coding one's own little extensions to Emacs, the argument being that few people actually do it. While it is probably true that only a minority of Emacs users learn enough Elisp to do it, I beg to differ — I think many more people *could* benefit from doing exactly this if only they knew how. There is this lovely quotation from one of rms' speeches [9], however, which comments on the issue:

> Multics Emacs proved to be a great success — programming new editing commands was so convenient that even the secretaries in his office started learning how to use it. They used a manual someone had written which showed how to extend Emacs, but didn't say it was a programming [task]. So the secretaries, who believed they couldn't do programming, weren't scared off. They read the manual, discovered they could do useful things and they learned to program.

Please consider this section as a kind of teaser which might hook you into Elisp programming.

3.1 Support for tildes

As a Polish TEX user, I tend to put a lot of tildes (hard spaces) in my files. (A Polish tradition is not to break a line after a one-letter word, and we have a few one-letter prepositions and conjunctions.) Of course, TEX has a solution to that: tildes (called "ties" in *The TEXbook*). Typing them manually is rather tedious, though. Happily, Emacs has you covered: there is a (built-in) package called "tildify", which replaces spaces after one-letter words (or in other places, since it is configurable). It is not even restricted to TEX — it can insert ` ` in HTML files, for example. It can do it dynamically while you type or after the fact on some portion of the file.

This is, however, not enough. I find myself editing other people's files on a regular basis, and oftentimes I need to insert a tie where a space was. In a regular editor this means navigating to the right place, deleting the space and inserting the tie. At one point I asked myself a question: how often do I need to have a tie next to a space? The answer is: *never*. Thus I decided to bind the tilde to a command which

```
(defun smart-tie ()
  "Delete any whitespace character(s),
then insert a tilde."
  (interactive)
  (delete-horizontal-space)
  (insert "~"))

(eval-after-load 'tex
  '(define-key TeX-mode-map "~" 'smart-tie))
```

Figure 2: The `smart-tie` Elisp source code

```
(add-hook 'TeX-mode-hook
  (lambda ()
    (font-lock-add-keywords nil
     '(("~" . 'font-latex-sedate-face)))))
```

Figure 3: A snippet making ties gray

starts by deleting any whitespace around point and only then inserting a tie (figure 2).

Emacs commands are just Elisp functions (defined with `defun`), which contain an `(interactive)` call at the beginning. (Notice that before that, there is a docstring. While not mandatory, it is a good practice to include it in all Elisp functions.) Notice that one of the reasons coding simple Emacs commands like this is, well, simple, is that you can just write down the things you press in order to make a similar edit manually, then check what commands are run by these key-presses, and just make a function out of them. It is even possible to have Emacs do it for you — you can record a so-called *keyboard macro*, performing some editing functions by hand, and let a suitable command generate an Elisp function mimicking your actions.

(I am not going to explain Elisp syntax in this article. I think much of it is pretty self-explanatory for anyone into programming, and for those new to it, the book [2] is a nice general introduction to both programming and Emacs Lisp.)

One last thing connected with ties is legibility. While I appreciate the fact that a (LA)TEX source file is plain text and hence I can see exactly where a hard space is, lots of them make the text less readable. I would prefer if they were gray instead of black (I use a dark-on-light default color theme). This is not a problem for Emacs. I put the snippet from figure 3 in my init file, and from now on all my tildes are displayed in gray. Here you can see a hook in action. `TeX-mode-hook` contains a list of functions called when turning any TEX-like mode on. We add to the hook an anonymous function (introduced with the `lambda` macro) which adds the `~` character to the "keywords" recognized by the font-locking machinery.

```
(defun smart-self-insert-punct (count)
  "If COUNT=1 and the point is after
a space, insert the relevant character
before any spaces."
  (interactive "p")
  (if (and (= count 1)
           (eq (char-before) ?\s))
      (save-excursion
        (skip-chars-backward " ")
        (self-insert-command 1))
  (self-insert-command count)))

(eval-after-load
  'tex
  '(define-key TeX-mode-map
              ","
              'smart-self-insert-punct))
```

Figure 4: The `smart-self-insert-punct` Elisp code

3.2 Smart commas

A common mistake is to forget a comma where it is needed; a copyeditor has to insert a lot of these. Since many navigation commands land the point at the beginning of some word, I always had to press the left arrow and then insert a comma. And then it struck me that I virtually *never* need a comma *after* a space between words, so why not automate this? And thus I wrote a short command called `smart-self-insert-punct` (see figure 4), which detects whether the point is after a space, and if yes, backs up first before entering the character used to issue the command.

This code is more or less self-explanatory (at least when you get accustomed to the Lisp prefix notation — for instance, to check for equality of two numbers `a` and `b`, you write `(= a b)`), but two things are probably worth mentioning. First of all, the `(interactive "p")` part performs some tricks so that `count` is one unless the user presses something like `C-u` ⟨*number*⟩ before issuing the above command. This is called a *prefix argument* and serves as a repeat count for many commands. Then, we have the very useful `save-excursion` form, which remembers the position of the point, performs the code given and returns the point to its previous position. (You usually do not expect the point to jump around when Emacs does something, and Emacs can do a lot of things — like spell-checking, for instance — even without the user doing anything.)

3.3 Converting `\cites`

As an editor of *Wiadomości Matematyczne* I often receive a paper with lots of citations done wrong. Many times the author says something like

```
see papers~\cite{A} and~\cite[p.~12]{B}.
```

```
(defun skip-cite-at-point ()
  "Move point to the end of the \\cite
at point."
  (when (looking-at "\\\\cite")
    (forward-char 5)
    (cond ((= (char-after) ?\[)
           (forward-sexp 2))
          ((= (char-after) ?\{)
           (forward-sexp)
           (when (and (not (eobp))
                      (= (char-after) ?*))
             (forward-char)
             (forward-sexp)))
          (t (error
              "Malformed \\cite")))))

(defun cites-to-citelist ()
  "Convert region to a \\citelist command.
All \\cite's are preserved and things
between them deleted.  This command will
be fooled by things like \"\\\\cite\"."
  (interactive)
  (if (use-region-p)
      (let
          ((end (copy-marker (region-end))))
        (goto-char (region-beginning))
        (insert "\\citelist{")
        (while (< (point) end)
          (skip-cite-at-point)
          (delete-region
           (point)
           (if (search-forward "\\cite" end t)
               ((progn )
                (backward-char 5)
                (point))
             end)))
        (insert "}"))
    (message "Region not active")))
```

Figure 5: The `cites-to-citelist` command

Since we use AMSrefs, this should be converted to something along the lines of

```
see papers~\citelist{%
  \cite{A}\cite{B}*{p.~12}}.
```

Hence I wrote another simple Elisp command, called `cites-to-citelist` (see figure 5), which performs the conversion for me. (It does not perform the whole job, i.e., it leaves the optional argument in brackets. This is not a huge problem, since I have another command to convert it to the AMSrefs syntax.) These commands are actually more complicated. I will not explain them in full, but let me highlight a few key points. (If you are interested in learning the details, you can use [4] and/or install Emacs and look at the docstrings of all the functions called in

the above code.) Again, let me emphasize that writing a `skip-cite-at-point` function is easier than it might seem, since it mimics the operations you (as a user) would perform to move the point past the `\cite` LaTeX macro: first, you check whether you are actually on it, then move by five characters, then move forward past the part(s) enclosed in brackets/braces. Also, in the `cites-to-citelist` function, we utilize the *region*, which is the Emacs term for the selection.

4 Conclusion

As you could hopefully see, Emacs works extremely well as a (LA)TeX editor. There are three reasons for that. First and foremost, it is an excellent general-purpose editor, with a simple TeX mode included. Secondly, there is the AUCTeX package, which is a robust tool, still under active development, and numerous other packages, like RefTeX, pdf-tools and Org-mode, which make the experience even better. The third reason is that Emacs truly delivers on its promise to be *extensible, customizable, self-documenting*, and automating repetitive tasks is fairly easy. If you are currently using TeXworks or even Vim (or any other TeX editor — there are so many of them), do yourself a favor and try out Emacs. You might stay in it for your whole life!

If you want to learn more about Emacs, you can install it and start with the built-in tutorial and proceed to at least skimming the manual. There is also a reference card included in the distribution, and others available on the Internet. A very good source of tips for using (though not programming) Emacs is Mickey Petersen's book *Mastering Emacs* [6]. A good source of useful information is *Planet Emacsen* [7], an Emacs blog aggregator. You can ask all sorts of Emacs-related questions on the official mailing list [5]. If you want to start your own adventure with Elisp, definitely start with Robert Chassell's *An Introduction to Programming in Emacs Lisp* [2]. Finally, let me mention a (now dormant) project of mine of writing a modern sequel to Chassell's book, which I hope to revive this year; I will surely post updates to it on my blog at `http://mbork.pl/Content_AND_Presentation`.

Acknowledgments

I would like to thank my friends from the `gust-l` and `help-gnu-emacs` mailing lists for their valuable input and many suggestions, and I am indebted to the editors for their excellent proofreading job.

Bibliography

[1] Marcin Borkowski, *Ten years of work in Wiadomości Matematyczne — an adventure with LaTeX and Emacs hacking*, TUGboat **38** (2017), no. 2, 255–263. `https://tug.org/TUGboat/tb38-2/tb119borkowski.pdf`.

[2] Robert J. Chassell, *An Introduction to Programming in Emacs Lisp*, 3rd ed., GNU Press, 2006. Bundled with Emacs source code and available in the Emacs Info documentation system. `https://www.gnu.org/software/emacs/manual/eintr`.

[3] Bob Glickstein, *Writing GNU Emacs Extensions*, 1st ed., O'Reilly Media, 1997.

[4] Bil Lewis et al., *GNU Emacs Lisp Reference Manual*. Bundled with Emacs source code and available in the Emacs Info documentation system. `https://www.gnu.org/software/emacs/manual/elisp`.

[5] *help-gnu-emacs*: *Users list for the GNU Emacs text editor*, `https://lists.gnu.org/mailman/listinfo/help-gnu-emacs`.

[6] Mickey Petersen, *Mastering Emacs*, v2, 2016. Available at `https://www.masteringemacs.org/book`.

[7] *Planet Emacsen*, `http://planet.emacsen.org/`.

[8] Richard M. Stallman et al., *GNU Emacs Manual*. Bundled with Emacs source code and available in the Emacs Info documentation system. `https://www.gnu.org/software/emacs/manual/emacs`.

[9] Richard M. Stallman, *My Lisp Experiences and the Development of GNU Emacs* (2002), available at `https://www.gnu.org/gnu/rms-lisp.html`.

◇ Marcin Borkowski
Faculty of Mathematics
and Computer Science
Adam Mickiewicz University
ul. Umultowska 87
61-614 Poznań, Poland
`mbork (at) amu dot edu dot pl`
`http://mbork.pl`

Editor's note: As it happens, I (Karl), like Marcin, work in Emacs, but my environment is set up completely differently from his. After comparing notes, Marcin and I thought it might be interesting to briefly describe mine as well, as an example of Emacs's extreme customizability and extensibility. All my changes are done at the Elisp level.

Some 35 years ago when I started using Emacs, my basic idea is to eradicate editing modes altogether. No `tex-mode`, no `c-mode`, etc. Keystrokes mean the same thing no matter what's being edited. I eliminate all fontification and colorization. Those are just distractions for me; I want to focus on the text.

I've also rebound nearly every key, and created hundreds of new bindings and many simple functions, so that I can do more things with less effort. For instance: save all buffers and run (what's normally) `M-x compile` with one keystroke. I typically do this dozens of times a day (I use `make` for essentially all building, e.g., running TeX). I read mail inside Emacs, use shell buffers (inside Emacs) for working locally, ssh buffers (inside Emacs) for working remotely, besides logging in remotely ... to run Emacs.

I primarily still use Emacs 21.[34], in terminal mode (not X mode), because (a) the Unicode support in new releases is painful for me when editing *TUGboat* papers in other encodings (autorecognition of encodings doesn't always work), or which use characters not in my favorite font. Just give me the bytes! (b) The changing of interfaces at every level, with no easy way back to previous behavior, that the Emacs developers have engaged in is too time-consuming for me to keep up with, especially when there is no significant benefit to the new versions in my environment. ◇

Tutorial: Using external C libraries with the LuaTEX FFI

Henri Menke

Abstract

The recent 1.0.3 release of LuaTEX introduced an FFI library (Foreign Function Interface) with the same interface as the one included by default in the LuaJIT interpreter. This allows for interfacing with any external library which adheres to the C calling convention for functions, which is pretty much everything. In this tutorial I will present how to interface with the GNU Scientific Library (GSL) to calculate integrals numerically. As a showcase I will plot a complete Fermi-Dirac integral using the `pgfplots` package. To understand this article, the reader should have good knowledge of the Lua and C programming languages and a basic understanding of the C memory model.

1 The FFI library

Lua is known for its rich C API which allows interfacing with system libraries in a straightforward fashion. The workflow for that is always the same: Write a function in C which fetches the arguments from the stack of the Lua interpreter and converts them into fixed C types. Using the fixed-type variables call the library function and receive a result, either as a return value or as an output argument. The result has to be converted back to a Lua type and pushed onto the stack of the Lua interpreter. Then hand the control back to the interpreter.

As we can already see, this recipe involves writing a lot of code, most of which is pure boilerplate. Wouldn't it be great if there was something which would just do all the type conversion work for us? And indeed there is, the FFI [3, 5, 8]. The concept of a Foreign Function Interface is not exclusive to Lua and also exists in other languages, e.g. with the `ctypes` library for Python.

Different FFIs have different ways of binding library functions. The Lua FFI chooses to parse plain C declarations. The advantage of this is that when interfacing with C libraries, you can copy and paste function prototypes from corresponding header files. Of course, the disadvantage is that for non-C libraries you have to come up with those prototypes yourself, which is not always an easy task. The FORTRAN language, for example, does not use the C-style *call by value* convention but always uses *call by reference*; that is to say, all types from a C function prototype would have to be converted to pointer types.

Thanks to Hans Hagen for very useful discussions.

2 The GNU Scientific Library

The GNU Scientific Library (GSL) [2] is a software library for scientific computing, implementing a broad range of algorithms. A complete list of algorithms is far too long to be presented here, and beyond the scope of this tutorial. We will only deal with the numerical integration routines here.

The numerical integration routines in the GSL are based on algorithms from the QUADPACK [9] package for adaptive Gauss-Legendre integration. In essence, each of the functions computes the integral

$$I = \int_a^b f(x)w(x)\,dx \tag{1}$$

where $w(x)$ is a weight function. We will be focussing only on the case where the weight function $w(x) = 1$. Since an integral cannot be solved exactly by a computer, the user has to provide error bounds to indicate convergence.

3 Interfacing with the GSL

The first thing to do when we want to interface with an external library is load the FFI Lua module and use it to load the shared library of interest into memory.

```
local ffi = require("ffi")
local gsl = ffi.load("gsl")
```

3.1 C declarations

Next we have to add all the C declarations which are important for us. Let us first have a look over the code and then discuss why I wrote things the way they are.

```
ffi.cdef[[
typedef double(*gsl_cb)(double x, void *);

typedef struct {
  gsl_cb F;
  void *params;
} gsl_function;

typedef void gsl_integration_workspace;

gsl_integration_workspace *
 gsl_integration_workspace_alloc(size_t n);

void gsl_integration_workspace_free(
  gsl_integration_workspace * w);

int gsl_integration_qagiu(
  gsl_function *f,
  double a, double epsabs, double epsrel,
  size_t limit,
  gsl_integration_workspace *workspace,
  double *result, double *abserr);
]]
```

The first declaration introduces a new type, which I call `gsl_cb`, which stands for GSL callback. It is a pointer to a function which takes a floating point number and a void pointer and returns another floating point number. In reality, this function pointer will point to a Lua function representing the integrand, i.e. $f(x)$ in Eq. 1. We can ignore the unnamed second argument (`void *`) here because this is only relevant for the C interface of the GSL but we still have to declare it.

The next declaration is another type declaration, this time with the name `gsl_function`. It is a structure containing two values; the first is the function pointer to the integrand `F`, the second a pointer to some memory where parameters could be located. In our case we will not use the `params` field but we nevertheless have to declare it. What is very important is that the order of the fields in the structure is *exactly the same* as in the C header file. Otherwise the memory alignment of the field will be off and a segmentation fault will occur.

The last type declaration is for the identifier `gsl_integration_workspace`, which I simply make it an alias for `void`. Looking in the C header file of the GSL, we find that `gsl_integration_workspace` is defined as a structure with several fields, so why do we not declare those fields? The reason is simple: we don't care. As you will see we do not access any fields of `gsl_integration_workspace` from the Lua level and the GSL library already knows what the fields are. Therefore I decided to make `gsl_integration_workspace` *opaque*.

The next three declarations are all function declarations which are straight copies from the header file: `gsl_integration_workspace_alloc` allocates enough memory to perform integration using `n` subintervals; `gsl_integration_workspace_free` releases that memory back to the system; and the third function declaration, `gsl_integration_qagiu`, is the actual integration routine. It computes the integral of the function `f` over the semi-infinite interval from `a` to ∞ with the desired absolute and relative error limits `epsabs` and `epsrel` using at most `limit` subintervals which have been previously allocated in `workspace`. The final approximation and the corresponding absolute error are returned in `result` and `abserr` [10].

3.2 Lua interface

Now that we've declared all of the library functions it is time that we integrate this with Lua. To this end we write a function which nicely encapsulates all the lower level structure. The function is named `gsl_qagiu` and takes as parameters a (Lua) function `f` (which takes one argument), the lower limit of the integral `a`, and three optional arguments, the absolute error `epsabs`, the relative error `epsrel`, and the maximum number of subintervals `N`.

```
local gsl_f  = ffi.new("gsl_function")
local result = ffi.new("double[1]")
local abserr = ffi.new("double[1]")

function gsl_qagiu(f,a,epsabs,epsrel,N)
  local N = N or 50
  local epsabs = epsabs or 1e-8
  local epsrel = epsrel or 1e-8

  gsl_f.F = ffi.cast("gsl_cb",f)
  gsl_f.params = nil

  local w =
     gsl.gsl_integration_workspace_alloc(N)

  gsl.gsl_integration_qagiu(gsl_f, a,
     epsabs, epsrel, N,
     w, result, abserr)

  gsl.gsl_integration_workspace_free(w)
  gsl_f.F:free()

  return result[0]
end
```

We start by defining some local variables outside the function for better performance. We instantiate a new value of type `gsl_function` and two arrays of length one using the `ffi.new` method.

After processing the optional arguments, we set the fields `F` and `params`. This is where it gets interesting. Recall that the type of the field `F` is a pointer to a function which takes two arguments. Even though the Lua function `f` only takes one argument we can use it directly, because of the way Lua deals with optional arguments. If the number of arguments is less than the number of parameters passed to the function call, all the additional parameters are simply dropped. The only problem that we have left is that this is a Lua function, not a C function. To this end we use `ffi.cast` to cast the Lua function into a C function. It can also be converted implicitly by simply assigning `f`, but then it is less clear what is going on. At this point it is very important that the types of the arguments and the return value match, otherwise we will run into memory problems. Because the field `params` is unused we simply set it to the null pointer by assigning `nil`. (We could probably leave it unset but that is bad practice. Always initialize your variables!)

The result and the absolute error of the integration are returned as output arguments from the GSL

function, i.e. the variables have to have pointer type. The easiest way to create a pointer to a value is by creating an array of length one of the desired type, which we already did outside the function. Arrays can be implicitly cast into pointers but at the same time live on the stack, so we do not have to worry about heap allocation and deallocation.

Next we use the previously declared functions to first allocate a workspace structure of sufficient size, then call the integration function with all of our arguments, releasing the workspace memory back to the system. You might notice that not all of the variables in the call to the integration routine have been created using `ffi.new`. This is indeed not necessary because the FFI will try to convert Lua values to native C types for you implicitly. Roughly speaking, you only have to use `ffi.new` for non-fundamental types or arrays.

There is one last subtlety to take care of. The library function to which we passed the function pointer is allowed to store that pointer for later use. Therefore this pointer will not decrease its reference count after exiting the function and therefore can never be garbage collected. We are probably not going to call this function so many times that this memory leak will have a huge impact but it is certainly good practice to release resources on exit, so we indicate to the garbage collector that this pointer can be cleaned up by calling its `free()` method.

Finally we return the result which is stored in the first element of the array. Note that C uses zero-based indexing.

3.3 Usage in pgfplots

So far we have only been implementing some kind of abstract skeleton for numerical integration. Now it is definitely time to actually use it. To this end we will plot the following complete Fermi-Dirac integral:

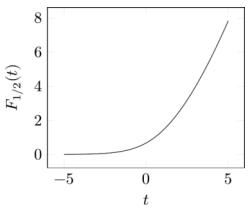
$$F_{1/2}(t) = \int_0^\infty \frac{x^{1/2}}{e^{x-t}+1} dx. \qquad (2)$$

What we will do now is call the `gsl_qagiu` routine with the integrand as the first argument and the lower limit as the second argument. Because we want to obtain the result of the integration in TEX we do not return the result of the integration but feed it back to TEX using `tex.sprint`.

```
function F_one_half(t)
  tex.sprint(gsl_qagiu(function(x)
      return math.sqrt(x)/(math.exp(x-t)+1)
    end, 0))
end
```

The last thing to do is plot this function using `pgfplots`. In the following I use ConTEXt syntax but the TEX and LATEX syntax is very similar. It should be noted though, that for FFI to work in LATEX, the `--shell-escape` option has to be enabled, because these operations are considered unsafe. First of all we need to tell Ti*k*Z about the Lua function. We do this using `declare function` and simply calling the Lua function with the argument. (A LATEX user would use `\directlua` instead of `\ctxlua`.) There is still a slight problem. The `pgfplots` package uses its own representation for floating point numbers, called `fpu` [1], which is not compatible with Lua. There are ways to work around this (see the Appendix), but the simplest solution for the moment is simply turning off the `fpu` for this plot.

```
\starttikzpicture
 [declare function={
  F_one_half(\t) = \ctxlua{F_one_half(\t)};
 }]
 \startaxis[
  use fpu=false, % very important!
  width=6cm,
  no marks,
  samples=101,
  xlabel={$t$},
  ylabel={$F_{1/2}(t)$},
 ]
  \addplot{F_one_half(x)};
 \stopaxis
\stoptikzpicture
```



4 Conclusion

The availability of FFI in LuaTEX takes document processing to a completely new level. The possibility to interface with native C libraries allows for tasks which were previously intractable, such as the numerical integration in this tutorial. This article was inspired by a question asked on Stack Exchange, where a minimal working example of the techniques presented here can be found [7].

Another example would be the conversion of an image from SVG format to PDF without the generation of intermediate files, as I demonstrated in [6] using the Cairo and Rsvg-2 libraries.

Finally, Aditya Mahajan published an article on his ConTEXt blog on how to interface the Julia programming language with LuaTEX via the FFI [4].

5 Appendix

During the preparation of this manuscript I was made aware, by Aditya Mahajan, that the approach of turning off the `fpu` is not always a viable workaround; it can fail, for instance when trying to plot in logscale. Therefore one has to convert the function argument from `fpu` float to Lua number and the result from

Lua number to fpu float. Fortunately PGF provides macros to facilitate this conversion. Using those one can declare the function from the main text as follows:

```
\pgfmathdeclarefunction{F_one_half}{1}{%
  \pgfmathfloatparsenumber{%
  \ctxlua{
    F_one_half(\pgfmathfloatvalueof{#1})
  }%
 }%
}
```

One does not necessarily have to rely on the macro level here. As of version 3, the PGF package comes with a Lua backend for function evaluations which provides parser functions for fpu types. With this, one could adapt the Lua function from the main text as follows:

```
local plf = require"pgf.luamath.functions"

function F_one_half(t)
  local t = plf.tonumber(t)
  local result = gsl_qagiu(function(x)
     return math.sqrt(x)/(math.exp(x-t)+1)
    end, 0))
  tex.sprint(plf.toTeXstring(result))
end
```

References

[1] Christian Feuersänger. Floating point unit library. https://ctan.org/pkg/pgf, 2015.

[2] M. Galassi et al. *GNU Scientific Library Reference Manual*. Network Theory Ltd., third edition, 2009.

[3] Hans Hagen, Luigi Scarso, and Taco Hoekwater. LuaTeX 1.0.3 announcement. https://tug.org/pipermail/luatex/2017-February/006345.html, 2017.

[4] Aditya Mahajan. Interfacing LuaTEX with Julia. https://adityam.github.io/context-blog/post/interfacing-with-julia/, 2017.

[5] James R. McKaskill. LuaFFI. https://github.com/jmckaskill/luaffi, 2010–2013.

[6] Henri Menke. Answer to 'How to include SVG diagrams in LaTeX?' https://tex.stackexchange.com/a/408014, 2017.

[7] Henri Menke. Answer to 'Plot complete Fermi-Dirac integral in Lualatex'. https://tex.stackexchange.com/a/403794, 2017.

[8] Mike Pall. LuaJIT: FFI library. http://luajit.org/ext_ffi.html, 2005–2017.

[9] R. Piessens, E. de Doncker-Kapenga, C.W. Überhuber, and D.K. Kahaner. *Quadpack: A Subroutine Package for Automatic Integration*. Springer, 1983.

[10] The GSL Team. GNU Scientific Library: Numerical integration. https://www.gnu.org/software/gsl/doc/html/integration.html, 1996–2017.

◇ Henri Menke
9016 Dunedin
New Zealand
henrimenke (at) gmail dot com

Executing TEX in Lua: Coroutines

Hans Hagen

Much of the Lua code in ConTEXt originates from experiments. When it survives in the source code it is probably used, waiting to be used or kept for educational purposes. The functionality that we describe here has already been present for a while in ConTEXt, but improved a little starting with LuaTEX 1.08 due to an extra helper. The code shown here is generic and not used in ConTEXt as such.

Say that we have this code:

```
for i=1,10000 do
    tex.sprint("1")
    tex.sprint("2")
    for i=1,3 do
        tex.sprint("3");tex.sprint("4")
        tex.sprint("5")
    end
    tex.sprint("\\space")
end
```

When we call `\directlua` with this snippet we get some 30 pages of `12345345345`. The printed text is saved till the end of the Lua call, so basically we pipe some 170.000 characters to TEX that get interpreted as one paragraph.

Now imagine this:

```
\setbox0\hbox{xxxxxxxxxxx} \number\wd0
```

which gives 4461336. If we check the box in Lua:

```
tex.sprint(tex.box[0].width)
tex.sprint("\\enspace")
tex.sprint("\\setbox0\\hbox{!}")
tex.sprint(tex.box[0].width)
```

the result is `4461336 4461336`, which is not what you would expect at first sight. However, if you consider that we just pipe to a TEX buffer that gets parsed after the Lua call, it will be clear that the reported width is the width that we started with. It will work all right if we say:

```
tex.sprint(tex.box[0].width)
tex.sprint("\\enspace")
tex.sprint("\\setbox0\\hbox{!}")
tex.sprint("\\directlua{\
             tex.sprint(tex.box[0].width)}")
```

because now we get: `4461336 443625`. It's not that complex to write some support code that makes this more convenient. This can work out quite well but there is a drawback. If we use this code:

```
print(status.input_ptr)
tex.sprint(tex.box[0].width)
tex.sprint("\\enspace")
tex.sprint("\\setbox0\\hbox{!}")
tex.sprint("\\directlua{print(status.input_ptr)\
    tex.sprint(tex.box[0].width)}")
```

Here we get `6` and `7` reported. You can imagine that when a lot of nested `\directlua` calls happen, we can get an overflow of the input level or (depending on what we do) the input stack size. Ideally we want to do a Lua call, temporarily go to TEX, return to Lua, etc. without needing to worry about nesting and possible crashes due to Lua itself running into problems. One charming solution is to use so-called coroutines: independent Lua threads that one can switch between — you jump out from the current routine to another and from there back to the current one. However, when we use `\directlua` for that, we still have this nesting issue and what is worse, we keep nesting function calls too. This can be compared to:

```
\def\whatever{\ifdone\whatever\fi}
```

where at some point `\ifdone` is false so we quit. But we keep nesting when the condition is met, so eventually we can end up with some nesting related overflow. The following:

```
\def\whatever{\ifdone\expandafter\whatever\fi}
```

is less likely to overflow because there we have tail recursion which basically boils down to not nesting but continuing. Do we have something similar in LuaTEX for Lua? Yes, we do. We can register a function, for instance

```
lua.get_functions_table()[1]
  = function() print("Hi there!") end
```

and call that one with:

```
\luafunction 1
```

This is a bit faster than calling a function like:

```
\directlua{HiThere()}
```

which can also be achieved by

```
\directlua{print("Hi there!")}
```

which sometimes can be more convenient. Anyway, a function call is what we can use for our purpose as it doesn't involve interpretation and effectively behaves like a tail call. The following snippet shows what we have in mind:

```
tex.routine(function()
    tex.sprint(tex.box[0].width)
    tex.sprint("\\enspace")
    tex.sprint("\\setbox0\\hbox{!}")
    tex.yield()
    tex.sprint(tex.box[0].width)
end)
```

We start a routine, jump out to TEX in the middle, come back when we're done and continue. This gives us: 4461336 218508, which is what we expect.

This mechanism permits efficient (nested) loops like:

```
tex.routine(function()
    for i=1,10000 do
        tex.sprint("1")
        tex.yield()
        tex.sprint("2")
        tex.routine(function()
            for i=1,3 do
                tex.sprint("3")
                tex.yield()
                tex.sprint("4")
                tex.yield()
                tex.sprint("5")
            end
        end)
        tex.sprint("\\space")
        tex.yield()
    end
end)
```

We do create coroutines, go back and forwards between Lua and TEX, but avoid memory being filled up with printed content. If we flush paragraphs (instead of e.g. the space) then the main difference is that instead of a small delay due to the loop unfolding in a large set of prints and accumulated content, we now get a steady flushing and processing.

However, we can still have an overflow of input buffers because we still nest them: the limitation at the TEX end has moved to a limitation at the Lua end. How come? Here is the code that we use:

```
local stepper = nil
local stack   = { }
local fid     = 0xFFFFFF
local goback  = "\\luafunction"..fid.."\\relax"

function tex.resume()
    if coroutine.status(stepper) == "dead" then
        stepper = table.remove(stack)
    end
    if stepper then
        coroutine.resume(stepper)
    end
end

lua.get_functions_table()[fid] = tex.resume

function tex.yield()
    tex.sprint(goback)
    coroutine.yield()
    texio.closeinput()
end

function tex.routine(f)
    table.insert(stack,stepper)
    stepper = coroutine.create(f)
    tex.sprint(goback)
end
```

The `routine` creates a coroutine, and `yield` gives control to TEX. The `resume` is done at the TEX end when we're finished there. In practice this works fine and when you permit enough nesting and levels in TEX then you will not easily overflow.

When I picked up this side project and wondered how to get around it, it suddenly struck me that if we could just quit the current input level then nesting would not be a problem. Adding a simple helper to the engine made that possible (of course figuring it out took a while):

```
local stepper = nil
local stack   = { }
local fid     = 0xFFFFFF
local goback  = "\\luafunction"..fid.."\\relax"

function tex.resume()
    if coroutine.status(stepper) == "dead" then
        stepper = table.remove(stack)
    end
    if stepper then
        coroutine.resume(stepper)
    end
end

lua.get_functions_table()[fid] = tex.resume

if texio.closeinput then
    function tex.yield()
        tex.sprint(goback)
        coroutine.yield()
        texio.closeinput()
    end
else
    function tex.yield()
        tex.sprint(goback)
        coroutine.yield()
    end
end

function tex.routine(f)
    table.insert(stack,stepper)
    stepper = coroutine.create(f)
    tex.sprint(goback)
end
```

The trick is in `texio.closeinput`, a recent helper and one that should be used with care. We assume that the user knows what she or he is doing. On an old laptop with a i7-3840QM processor running Windows 10 the following snippet takes less than 0.35 seconds with LuaTEX and 0.26 seconds with LuaJITTEX.

```
tex.routine(function()
  for i=1,10000 do
    tex.sprint("\\setbox0\\hpack{x}")
    tex.yield()
```

```
    tex.sprint(tex.box[0].width)
    tex.routine(function()
      for i=1,3 do
        tex.sprint("\\setbox0\\hpack{xx}")
        tex.yield()
        tex.sprint(tex.box[0].width)
      end
    end)
  end
end)
```

Say that we run the bad snippet:

```
for i=1,10000 do
    tex.sprint("\\setbox0\\hpack{x}")
    tex.sprint(tex.box[0].width)
    for i=1,3 do
        tex.sprint("\\setbox0\\hpack{xx}")
        tex.sprint(tex.box[0].width)
    end
end
```

This time we need 0.12 seconds in both engines. So what if we run this:

```
\dorecurse{10000}{%
    \setbox0\hpack{x}
    \number\wd0
    \dorecurse{3}{%
        \setbox0\hpack{xx}
        \number\wd0
    }%
}
```

Pure TEX needs 0.30 seconds for both engines but there we lose 0.13 seconds on the loop code. In the Lua example where we yield, the loop code takes hardly any time. As we need only 0.05 seconds more it demonstrates that when we use the power of Lua the performance hit of the switch is quite small: we yield 40,000 times! In general, such differences are far exceeded by the overhead: the time needed to typeset the content (which `\hpack` doesn't do), breaking paragraphs into lines, constructing pages and other overhead involved in the run. In ConTEXt we use a slightly different variant which has 0.30 seconds more overhead, but that is probably true for all Lua usage in ConTEXt, but again, it disappears in other runtime.

Here is another example:

```
\def\TestWord#1%
  {\directlua{
     tex.routine(function()
       tex.sprint("\\setbox0\\hbox{\\tttf #1}")
       tex.yield()
       tex.sprint(math.round
         (100 * tex.box[0].width/tex.hsize))
       tex.sprint(" percent of the hsize: ")
       tex.sprint("\\box0")
     end)
  }}

The width of next word is \TestWord {inline}!
```

The width of next word is 9 percent of the hsize: inline!

Now, in order to stay realistic, this macro can also be defined as:

```
\def\TestWord#1%
  {\setbox0\hbox{\tttf #1}%
   \directlua{
      tex.sprint(math.round(
         100 * tex.box[0].width/tex.hsize))
   } %
   percent of the hsize: \box0\relax}
```

We get the same result: "The width of next word is 9 percent of the hsize: `inline`!".

We have been using a Lua–TEX mix for over a decade now in ConTEXt, and have never really needed this mixed model. There are a few places where we could (have) benefitted from it and we might use it in a few places, but so far we have done fine without it. In fact, in most cases typesetting can be done fine at the TEX end. It's all a matter of imagination.

◇ Hans Hagen
Pragma ADE
http://pragma-ade.com

New rules for reporting bugs in the LaTeX core software (as maintained by the LaTeX Project)

Frank Mittelbach and The LaTeX Project Team

Abstract

Software has bugs and LaTeX unfortunately is no exception. If somebody encounters a bug then it helps if that bug gets reported to the right people so that the bug can be resolved (or a workaround documented or whatever is most appropriate). The problem is to know to whom to report the bug. For this the `latexbug` package has been developed to help in addressing the right group of maintainers.

The LaTeX Project Team maintains a bug database for its own code base (which consists of the LaTeX kernel and some packages that have been written by people in the LaTeX Project Team).

In this article we describe how to report bugs in the core LaTeX software or search through already known issue reports in that database. The article also explains where to find the development version of LaTeX if that ever becomes necessary.

Thank you for taking the time to report a bug and prepare a test file showing it!

Contents

1 The LaTeX kernel sources

LaTeX (or more precisely LaTeX 2_ε, the current standard) is part of every major TeX distribution, e.g., TeX Live, MiKTeX, or MacTeX to name a few. The official releases of LaTeX are all published on CTAN (the Comprehensive TeX Archive Network) where they can be downloaded if necessary and from there they usually find their way into the major distributions within a few days.

Until recently the LaTeX Project Team has maintained the development version of LaTeX 2_ε in an SVN (Subversion) repository with read-only access for the public from the LaTeX Project website. We have now switched to a Git repository[1] located at

`https://github.com/latex3/latex2e`

and from that browser page you can explore the files in the development version.

If necessary, the most recent (unreleased) development version can be downloaded from there in a `.zip` archive (roughly 25Mb) by using the appropriate button. If you are familiar with Git or SVN you can also clone the repository using the command line or your favorite Git frontend tool or alternatively do a checkout using an SVN tool.

1.1 A note on Git pull requests

Git repositories (somewhat in contrast to SVN ones) support widely distributed development and allow people to provide change sets that are made available through so-called *pull requests*, so that the maintainers of a program can "pull the suggested changes" into the main repository.

While we appreciate contributions, we think that for the core LaTeX software pull requests are usually not a good approach (unless the change has been previously discussed and agreed upon).

The stability of LaTeX is very important and this means that changes to the kernel are necessarily very conservative. It also means that a lot of discussion has to happen before any changes are made. So if you do decide to post a pull request, please bear this in mind: we do appreciate ideas, but cannot always integrate them into the kernel and it is quite likely that we will have to reject updates made in this way.

If you want to discuss a possible contribution before (or instead of) making a pull request, we suggest you raise the topic first on the LATEX-L list (see links below) or drop a line to the team.

2 Policy on layout and interface deficiencies

Up front we should probably stress that "deficiencies" in the design of the standard document classes (`article`, `report` and `book`) as well as questionable but long established interface behavior of commands are things that we will normally not change, even if we can all agree that a different behavior or a different layout would have been a better choice. You are, of course, welcome to report issues in these areas, using the procedure explained below, but in all likelihood such reports will get suspended.

[1] Please note, that if you have previously bookmarked the old SVN repository you should update that bookmark to the new Git repository as the SVN repository is frozen and no longer up-to-date and will soon vanish!

The reason is that the kernel interfaces and the document classes have been used for many years in essentially all documents (even documents using different classes; these are often built upon the standard classes in the background) and thus such changes would break or at a minimum noticeably change nearly all existing documents.

3 The bug database for the LaTeX kernel and core packages

Throughout the last two decades the LaTeX Project Team has maintained a bug database using GNATS, a free software system from the FSF. While this has served us well in the past, it has its problems and with our move to Git-based source control its workflow no longer fits. We have therefore decided to switch to a new tracking system and the natural choice was to use the one already provided as part of the GitHub setup (the place where the sources are now hosted), namely the Issue Tracker.

Unfortunately, it is not possible for a number of reasons to automatically transfer the old bug reports to the new system so we are in a slightly awkward position that we have old bugs in one system and the new ones in another. Thus for searching through already reported bugs it is necessary to search two systems:

- GNATS for bugs reported before 2018;
- The Github Issue Tracker for LaTeX 2_ε for bugs reported in 2018 and later.

Over time we hope that the bugs listed in GNATS will all be only of historical interest, but right now it is probably helpful to look in both places (see links below) — sorry for that.

4 The `latexbug` package

So far we have talked about where to find the core LaTeX software and how to report issues with it. However, the LaTeX universe consists of several thousand contributed packages maintained by individuals all over the world. And if a bug happens in one of those packages it doesn't help anybody if it is dumped at the LaTeX Project's doorstep.

For one, we can't actually change other people's code even if we are able to identify the issue. Furthermore we are only a few people and simply do not have the bandwidth to analyze bugs in other people's work.

We have therefore written this little package called `latexbug` that should help in identifying the rightful addressee for a bug report. We ask that it be loaded in any test file intended to be sent to the LaTeX bug database as part of a bug report.

The package will determine if the test file is in a suitable state to be sent to us or if it should be modified first or if it should be sent to somebody else because the bug is (likely) to be in code not maintained by the LaTeX Project Team.

Bug reports sent to the LaTeX bug database without that prior verification are likely to get closed without being looked at at our end in the future.

4.1 The user interface

The interface is simple: the package has no options and doesn't define any new commands to be used.

All that is required is that the package be loaded as the very first step in the test file that shows the bug — in other words, before the line loading the `\documentclass`. For that reason it must be loaded using `\RequirePackage` instead of the usual `\usepackage` declaration that is used in the preamble of a document.

Thus, a bug report test file should look like this:

```
\RequirePackage{latexbug}
\documentclass{article}
  % preamble as necessary
  % (drop anything not needed, please)
\begin{document}
  % example showing the bug
  % (as short and concise as possible)
\end{document}
```

Of course, instead of `article` you may need to load a different standard class, but do not load a third-party class as we can't accept a bug that manifests itself only when using a class we don't maintain.

If the test file runs through (showing the bug) without any complaints for `latexbug` then the test file is ready to be sent to the LaTeX bug database. The procedure for uploading and the location is documented at

`https://www.latex-project.org/bugs/`

If `latexbug` does generate an error, however, then this error needs to be addressed first and then, depending on the resolution, the bug report may have to be sent to somebody else.

An error is generated if the test file makes use of third-party code that is not maintained by the LaTeX Project Team. For example, if your test document loads `array`, `geometry`, `footmisc` and `hyperref` you would see the following:

```
Package latexbug Error: Third-party file(s)

 This test file uses third-party file(s)
 ==============
 geometry.sty  ->  Hideo Umeki
```

```
 <latexgeometry [at] gmail [dot] com>
footmisc.sty
hyperref.sty  ->  Heiko Oberdiek
 https://github.com/ho-tex/hyperref/issues
==============
```

The `array` package is accepted as it is one of the core packages maintained by the LaTeX Project team but the other three are not. For `geometry` and `hyperref` we have maintainer info available, so we provide that, whereas for `footmisc` this information is missing. Thus, in that case you have to search for it yourself, if it turns out that the bug is related to that package.

The `latexbug` package then continues with advice to remove such third-party code from the file:

```
So you should contact the authors
of these files, not the LaTeX Team!
(Or remove the packages that load
them, if they are not necessary to
exhibit the problem).
```

If that is not possible, because the bug goes away if a package is removed, then the problem is (most likely) with this package and the bug report should be sent to the maintainer of that package and not to the LaTeX bug database.

To make life somewhat easier, `latexbug` will tell you the name of the maintainer (if we know it and have added it) and if possible also the canonical bug address for the package (as in the cases of `geometry` and `hyperref`). If we don't have that information, you need to find it out for yourself by looking at the package documentation.

There may be cases where third-party code is essential to exhibit a bug in core LaTeX code maintained by the LaTeX Team. The error text therefore finishes off with the following sentence:

```
If you think the bug is in core LaTeX
(as maintained by the LaTeX Team) but
these files are needed to demonstrate
the problem, please continue and mention
this explicitly in your bug report.
```

Please explain in detail your reasoning why you think this is the case as part of the bug report.

4.2 Bugs in `latexbug` itself

When a document is run through LaTeX it will load a number of files, and bug reports that are to be sent to the LaTeX Team should only load files that we maintain and not third-party packages. Testing this and giving some appropriate advice is the main task of the `latexbug` package.

The database inside `latexbug`, if you want to call it that, is simply a comma separated key value list consisting of file names = maintainer info. Most of the time the maintainer info is `us` (meaning we maintain it, so the file is fine) or `us*` (meaning it is an `expl3` package we maintain, so fine too, but should be reported in a different issue tracker) or `ignore` (meaning we do not maintain it, but it is a file that is likely to appear for one or the other reason and we should accept such a bug report nonetheless). We allow, for example, the use of `lipsum` or `blindtext` to help in making up a test file with a suitable amount of text. Also often useful is the package `etoolbox`, thus that is also silently accepted (aka ignored).

Any other file loaded in the bug report but not listed in the database will show up in the error listing flagged as "third-party" code that should be removed as explained above.

For a small number of popular third-party packages we have collected the name of the external maintainer and if available to us some extra information so that it is easier to send to the rightful addressee if you encounter a bug in such a third-party package. But to keep this manageable this is only done for a very small number of the 5000+ packages out there (though we might add a few more over time).

It is however not impossible that we missed one or another file that should have been listed as "maintained by us" but isn't and thus incorrectly generates an error. Another potential problem area is with the maintainer info we provide, as that might become invalid without being noticed.

If you run into one of those problems or notice an omission of that sort, please send us a bug report by opening an issue at the GitHub source of the package which is located at:

`https://github.com/latex3/latexbug`

Please note that the fact that a particular package is written by one of the members of the LaTeX Project team does not automatically mean that `latexbug` will classify it as a core LaTeX package. Many such packages will show up as "third-party" with the request to report the bug with the respective maintainer directly.

For example, `fontspec`, written by Will Robertson, has its own repository, so issues involving that package should normally be reported there and not with the LaTeX kernel, and `latexbug` will point you in the right direction.

5 Important links

`https://www.latex-project.org`
: Website of the LaTeX Project (official site for LaTeX and LaTeX3 development).

`https://www.latex-project.org/bugs`
: Page describing how to submit a bug report for core LaTeX. This should always contain the correct up-to-date links, etc.

`https://www.latex-project.org/latex3/code/#discussing-it`
: Page describing how to join the LaTeX Project discussion list and how to retrieve old posts.

`https://www.latex-project.org/cgi-bin/ltxbugs2html`
: Place to look for bugs reported prior to 2018.

`https://github.com/latex3/latex2e/issues`
: Place to search through bug reports from 2018 onwards and to open a new bug report ("New Issue") for core LaTeX 2_ε.

`https://github.com/latex3/latex3/issues`
: Place to open a bug report for issues involving LaTeX3 or `expl3` packages.

`https://github.com/latex3/latexbug`
: Home repository for the sources of the `latexbug` package. Also contains the ready-to-use package in case it is not in your distribution.

`https://ctan.org/pkg/latex-base`
: The LaTeX kernel sources on CTAN.

`https://ctan.org/pkg/required`
: CTAN home of LaTeX core packages that are required to be present in any distribution.

◇ Frank Mittelbach
Mainz, Germany
`frank.mittelbach (at) latex-project (dot) org`
`https://www.latex-project.org`

◇ The LaTeX Project Team
`latex-team (at) latex-project (dot) org`
`https://www.latex-project.org`

LaTeX News

Issue 28, April 2018

Contents

A new home for LaTeX 2ε sources

In the past the development version of the LaTeX 2ε source files has been managed in a Subversion source control system with read access for the public. This way it was possible to download in an emergency the latest version even before it was released to CTAN and made its way into the various distributions.

We have recently changed this setup and now manage the sources using Git and placed the master sources on GitHub at

`https://github.com/latex3/latex2e`

where we already store the sources for expl3 and other work. As before, direct write access is restricted to LaTeX Project Team members, but everything is publicly accessible including the ability to download, clone (using Git) or checkout (using SVN). More details are given in [1].

Bug reports for core LaTeX 2ε

For more than two decades we used GNATS, an open source bug tracking system developed by the FSF. While that has served us well in the past it started to show its age more and more. So as part of this move we also decided to finally retire the old LaTeX bug database and replace it with the standard "Issue Tracker" available at Github.

The requirements and the workflow for reporting a bug in the core LaTeX software is documented at

`https://www.latex-project.org/bugs/`

and with further details also discussed in [1].

UTF-8: the new default input encoding

The first TeX implementations only supported reading 7-bit ASCII files—any accented or otherwise "special" character had to be entered using commands, if it could be represented at all. For example to obtain an "ä" one would enter `\"a`, and to typeset a "ß" the command `\ss`. Furthermore fonts at that time had 128 glyphs inside, holding the ASCII characters, some accents to build composite glyphs from a letter and an accent, and a few special symbols such as parentheses, etc.

With 8-bit TeX engines such as pdfTeX this situation changed somewhat: it was now possible to process 8-bit files, i.e., files that could encode 256 different characters. However, 256 is still a fairly small number and with this limitation it is only possible to encode a few languages and for other languages one would need to change the encoding (i.e., interpret the character positions 0–255 in a different way). The first code points 0–127 were essentially normed (corresponding to ASCII) while the second half 128–255 would vary by holding different accented characters to support a certain set of languages.

Each computer used one of these encodings when storing or interpreting files and as long as two computers used the same encoding it was (easily) possible to exchange files between them and have them interpreted and processed correctly.

But different computers may have used different encodings and given that a computer file is simply a sequence of bytes with no indication for which encoding is intended, chaos could easily happen and has happened. For example, the German word "Größe" (height) entered on a German keyboard could show up as "GrT̈àe" on a different computer using a different encoding by default.

So in summmary the situation wasn't at all good and it was clear in the early nineties that LaTeX 2ε (that was being developed to provide a LaTeX version usable across the world) had to provide a solution to this issue.

The LaTeX 2ε answer was the introduction of the inputenc package [2] through which it is possible to

provide support for multiple encodings. It also allows to correctly process a file written in one encoding on a computer using a different encoding and even supports documents where the encoding changes midway.

Since the first release of LaTeX 2ε in 1994, LaTeX documents that used any characters outside ASCII in the source (i.e. any characters in the range of 128–255) were supposed to load inputenc and specify in which file encoding they were written and stored. If the inputenc package was not loaded then LaTeX used a "raw" encoding which essentially took each byte from the input file and typeset the glyph that happened to be in that position in the current font—something that sometimes produces the right result but often enough will not.

In 1992 Ken Thompson and Rob Pike developed the UTF-8 encoding scheme which enables the encoding of all Unicode characters within 8-bit sequences. Over time this encoding has gradually taken over the world, replacing the legacy 8-bit encodings used before. These days all major computer operating systems use UTF-8 to store their files and it requires some effort to explicitly store files in one of the legacy encodings.

As a result, whenever LaTeX users want to use any accented characters from their keyboard (instead of resorting to `\"a` and the like) they always have to use

```
\usepackage[utf8]{inputenc}
```

in the preamble of their documents as otherwise LaTeX will produce gibberish.

The new default

With this release, the default encoding for LaTeX files has been changed from the "fall through raw" encoding to UTF-8 if used with classic TeX or pdfTeX. The implementation is essentially the same as the existing UTF-8 support from `\usepackage[utf8]{inputenc}`.

The LuaTeX and XeTeX engines always supported the UTF-8 encoding as their native input encoding, so with these engines inputenc was always a no-op.

This means that with new documents one can assume UTF-8 input and it is no longer required to always specify `\usepackage[utf8]{inputenc}`. But if this line is present it will not hurt either.

Compatibility

For most existing documents this change will be transparent:

- documents using only ASCII in the input file and accessing accented characters via commands;
- documents that specified the encoding of their file via an option to the inputenc package and then used 8-bit characters in that encoding;
- documents that already had been stored in UTF-8 (whether or not specifying this via inputenc).

Only documents that have been stored in a legacy encoding and used accented letters from the keyboard *without* loading inputenc (relying on the similarities between the input used and the T1 font encoding) are affected.

These documents will now generate an error that they contain invalid UTF-8 sequences. However, such documents may be easily processed by adding the new command `\UseRawInputEncoding` as the first line of the file. This will re-instate the previous "raw" encoding default.

`\UseRawInputEncoding` may also be used on the command line to process existing files without requiring the file to be edited

```
pdflatex '\UseRawInputEncoding \input'  file
```

will process the file using the previous default encoding.

Possible alternatives are reencoding the file to UTF-8 using a tool (such as recode or iconv or an editor) or adding the line

```
\usepackage[⟨encoding⟩]{inputenc}
```

to the preamble specifying the ⟨*encoding*⟩ that fits the file encoding. In many cases this will be `latin1` or `cp1252`. For other encoding names and their meaning see the inputenc documentation.

As usual, this change may also be reverted via the more general latexrelease package mechanism, by speciying a release date earlier than this release.

BOM: byte order mark handling

When using Unicode the first bytes of a file may be a, so called, BOM character (byte order mark) to indicate the byte oder used in the file. While this is not required with UTF-8 encoded files (where the byte order is known) it is nevertheless allowed by the standard and some editors add that byte sequence to the beginning of a file. In the past such files would have generated a "Missing begin document" error or displayed strange characters when loaded at a later stage.

With the addition of UTF-8 support to the kernel it is now possible to identify and ignore such BOM characters even before `\documentclass` so that these issues will no longer be showing up.

A general rollback concept for packages and classes

In 2015 a rollback concept for the LaTeX kernel was introduced. Providing this feature allowed us to make corrections to the software (which more or less didn't happen for nearly two decades) while continuing to maintain backward compatibility to the highest degree.

In this release we have now extended this concept to the world of packages and classes which was not covered initially. As the classes and the extension packages

have different requirements compared to the kernel, the approach is different (and simplified). This should make it easy for package developers to apply it to their packages and authors to use when necessary.

The documentation of this new feature is given in an article submitted to *TUGboat* and also available from our website [3].

Integration of remreset and chngcntr packages into the kernel

With the optional argument to `\newcounter` LaTeX offers to automatically reset counters when some counter is stepped, e.g., stepping a `chapter` counter resets the `section` counter (and recursively all other heading counters). However, what was until now missing was a way to undo such a link between counters or to link two counters after they have been defined.

This can be now be done with `\counterwithin` and `\counterwithout`, respectively. In the past one had to load the chngcntr package for this. For the programming level we also added `\@removefromreset` as the counterpart of the already existing `\@addtoreset` command. Up to now this was offered by the remreset package.

Testing for undefined commands

LaTeX packages often use a test `\@ifundefined` to test if a command is defined. Unfortunately this had the side effect of *defining* the command to `\relax` in the case that it had no definition. The new release uses a modified definition (using extra testing possibilities available in ε-TeX). The new definition is more natural, however code that was relying on the side effect of the command being tested being defined if it was previously undefined may have to add `\let\`⟨*command*⟩`\relax`.

Changes to packages in the tools category

LaTeX table columns with fixed widths

Frank published a short paper in *TUGboat* [4] on producing tables that have columns with fixed widths. The outlined approach using column specifiers "`w`" and "`W`" has now been integrated into the array package.

Obscure overprinting with multicol fixed

A rather peculiar bug was reported on StackExchange for multicol. If the column/page breaking was fully controlled by the user (through `\columnbreak`) instead of letting the environment do its job and if then more `\columnbreak` commands showed up on the last page then the balancing algorithm was thrown off track. As a result some parts of the columns overprinted each other.

The fix required a redesign of the output routines used by multicol and while it "should" be transparent in other cases (and all tests in the regession test suite came out fine) there is the off-chance that code that hooked into the internals of multicol needs adjustment.

Changes to packages in the amsmath category

With this release of LaTeX a few minor issues with amsmath have been corrected.

Updated user's guide

Furthermore, `amsldoc.pdf`, the AMS user's guide for the amsmath package [5], has been updated from version 2.0 to 2.1 to incorporate changes and corrections made between 2016 and 2018.

References

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

[2] Frank Mittelbach: *LaTeX 2ε Encoding Interface — Purpose, concepts, and Open Problems*. Talk given in Brno June 1995. `https://www.latex-project.org/publications/`

[3] Frank Mittelbach: *A rollback concept for packages and classes*. Submitted to TUGboat. `https://www.latex-project.org/publications/`

[4] Frank Mittelbach: *LaTeX table columns with fixed widths*. In: TUGboat, 38#2, 2017. `https://www.latex-project.org/publications/`

[5] American Mathematical Society and The LaTeX3 Project: *User's Guide for the `amsmath` package* (Version 2.1). April 2018. Available from `https://www.ctan.org` and distributed as part of every LaTeX distribution.

TEX.StackExchange cherry picking: expl3

Enrico Gregorio

Abstract

We present some examples of macros built with expl3 in answer to users' problems presented on `tex.stackexchange.com` to give a flavor of the language and describe its possibilities. Topics include list printing, string manipulation, macro creation, and graphics.

1 Introduction

My first answer on TEX.SX using expl3, the programming language for the future LATEX3, appeared in November, 2011 and a month later I issued the first version of kantlipsum. As every regular of TEX.SX knows, I like to use expl3 code for solving problems, because I firmly believe in its advantages over traditional TEX and LATEX programming.

I'll present some selected answers I have given (sometimes with modified code), also making some comparisons with traditional coding. Some objections to expl3 may be justified: it's verbose, it needs to load a few thousand lines of code. Yes, it's verbose and in my opinion this is one of its strengths: I don't think that `\hb@xt@` is clearer and easier to interpret than `\hbox_to_wd:nn`. Loading a few thousand lines of code is done almost instantly on modern machines.

Using expl3 doesn't free the user from knowing something about *macro expansion*, because this is how TEX works to begin with, but a big advantage is that commonly used and often misunderstood `\expandafter` tricks are (almost) never needed.

Some acquaintance with the language is needed for reading this paper, but I think that having the `interface3` manual at hand would be sufficient for removing most doubts.

2 List printing

Our first toy problem is to define a macro `\ocamllist` that prints a list of items in the style of OCaml. Thus we want `\ocamllist{}` to print opening and closing brackets with a thin space in between, while `\ocamllist{aa,bb}` should print $[aa; bb]$.[1]

Such a command should be flexible enough to allow recursive calls. There are several possible solution and the one by Petr Olšák is, as usual, brilliant:

```
\def\ocamllist#1{\ocamllistA #1,,}
\def\ocamllistA#1,{[#1\ocamllistB}
\def\ocamllistB#1,{%
  \ifx,#1,#1]%
  \else
    ;#1\expandafter\ocamllistB
  \fi
}
$\ocamllist{}$\par
$\ocamllist{aa}$\par
$\ocamllist{aa,bb}$\par
$\ocamllist{aa,\ocamllist{bb,cc},dd}$
\end
```

If we try it, the output is almost as required; it only lacks the thin space:

[]
$[aa]$
$[aa; bb]$
$[aa; [bb; cc]; dd]$

Writing such code, however, requires mastery of the low-level TEX language. Can we do it without having to define three macros for doing such a thing? And, most important, using a more natural language? This is, of course, where expl3 comes into the picture.

```
\ExplSyntaxOn
\NewDocumentCommand{\ocamllist}{m}
 {
  [
  \clist_set:Nn \l_hongxu_ocamllist_clist {#1}
  \clist_if_empty:NTF \l_hongxu_ocamllist_clist
   { \, }
   { \clist_use:Nn \l_hongxu_ocamllist_clist {;} }
  ]
 }
\clist_new:N \l_hongxu_ocamllist_clist
\ExplSyntaxOff
```

Yes, one needs to learn a bunch of new names for the basic functions, but there are several advantages. For instance, suppose we want to extend the macro so it accepts a star variant for automatically sized fences and an optional command for manually choosing the size of the brackets. Very easy with expl3:

```
\ExplSyntaxOn
\NewDocumentCommand{\ocamllist}{sO{}m}
 {
  \IfBooleanTF{#1}{\left[}{\mathopen{#2[}}
  \clist_set:Nn \l_hongxu_ocamllist_clist {#1}
  \clist_if_empty:NTF \l_hongxu_ocamllist_clist
   { \, }
   { \clist_use:Nn \l_hongxu_ocamllist_clist {;} }
  \IfBooleanTF{#1}{\right]}{\mathclose{#2]}}
 }
\clist_new:N \l_hongxu_ocamllist_clist
\ExplSyntaxOff
```

Now the (silly) input

```
$\ocamllist{}$\par
$\ocamllist{aa}$\par
$\ocamllist[\Big]{aa,bb}$\par
$\ocamllist*{\sum\limits_{i=1}^n a_i,bb,cc}$\par
$\ocamllist[\big]{aa,\ocamllist{bb,cc},dd}$
```

will produce

[1] `https://tex.stackexchange.com/questions/360958/`

$[\,]$
$[aa]$
$\left[aa; bb\right]$
$\left[\sum_{i=1}^{n} a_i; bb; cc\right]$
$\left[aa; [bb; cc]; dd\right]$

What's happening? In the extended macro, `s` stands for an optional `*`, whose presence can be tested with `\IfBooleanTF` which does the necessary branching. The `O{}` bit specifies an optional argument with empty default value.

Now let's analyze the bulk of the macro. With `\clist_set:Nn` we set a variable (of type `clist`) to the specified argument. This is not just the same as doing a standard `\def`, because the input is 'normalized'; for instance, leading and trailing spaces in the items are removed. This would not be a problem here, as the macro is called in math mode, but it could be for macros called in text mode. With `\clist_use:Nn` the contents of the `clist` is delivered with the specified separator *between* items. Not adding a semicolon after the last item is cleverly done by the plain TEX macros above, while with expl3 we need not worry about it. It should also be clear that `\clist_if_empty:NTF` checks whether the variable contains an empty list or not.

For those readers who are unacquainted with expl3 syntax, let's recall the main facts. A *function* in the language has a name consisting of three parts:

1. a *module name*, here `hongxu`, that acts as a sort of "name space" indicator;
2. a *proper name*, which consists of any string of characters describing the function's action, with parts separated by underscores;
3. a *signature*, after the mandatory colon, that specifies the arguments the function expects.

When reading expl3 code, one can immediately parse the arguments to a function, because of the signature. The main argument types are

- `N`, for arguments consisting of a single token, usually a control sequence, but also a character;
- `n`, for standard braced arguments;
- `T` and `F`, for the true and false branch of a conditional function, but they're syntactically the same as `n`, so the actual arguments should be braced.

There are others, and we'll see some of them in action.

Names of variables follow a similar scheme; a name consists of

1. a *prefix*, which should be `l` (local), `g` (global) or `c` (constant);
2. a module name as with functions;
3. a proper name;
4. the variable's type.

Sticking to this convention helps in reading and debugging code.

A slightly different approach for this problem would be with

```
\seq_set_split:Nnn \l_hongxu_ocamllist_seq { , } {#1}
\seq_if_empty:NTF \l_hongxu_ocamllist_seq
 { \, }
 { \seq_use:Nn \l_hongxu_ocamllist_seq {;} }
```

Here we use another data type, namely `seq` (sequence); the first command splits the input at commas, removing leading and trailing spaces from the items. An input such as `\ocamllist{,}` would be treated differently by the two versions: with `clist` it would produce an empty `clist` because of normalization; with `seq` the sequence would have two items. The choice depends on the needs at hand.

2.1 Two-row matrix input

A similar problem is typesetting a two-row matrix with the entries specified as comma-separated pairs *firstrow*/*secondrow*,[2] so the input `1/5, 2/6` means the matrix $\begin{smallmatrix}1&2\\5&6\end{smallmatrix}$. Cleverly written recursive macros are possible, but here I'll present an expl3 version.

```
\documentclass{article}
\usepackage{amsmath}
\usepackage{xparse}
\setcounter{MaxMatrixCols}{20} % or maybe more

\ExplSyntaxOn
\NewDocumentCommand{\twolinematrix}{O{}m}
 {
  \twoline_matrix:nn { #1 } { #2 }
 }

\seq_new:N \l__twoline_i_seq
\seq_new:N \l__twoline_ii_seq

\cs_new_protected:Nn \twoline_matrix:nn
 {
  \seq_clear:N \l__twoline_i_seq
  \seq_clear:N \l__twoline_ii_seq
  \clist_map_function:nN { #2 } \twoline_add:n
  \begin{#1matrix}
  \seq_use:Nn \l__twoline_i_seq { & }
  \\
  \seq_use:Nn \l__twoline_ii_seq { & }
  \end{#1matrix}
 }
\cs_new:Nn \twoline_add:n
 {
  \__twoline_add:w #1 \q_stop
 }
\cs_new_protected:Npn \__twoline_add:w #1/#2 \q_stop
 {
  \seq_put_right:Nn \l__twoline_i_seq { #1 }
  \seq_put_right:Nn \l__twoline_ii_seq { #2 }
```

[2] https://tex.stackexchange.com/questions/393053/

```
 }
\ExplSyntaxOff
\begin{document}
\[
\twolinematrix{1/6, 2/7, 3/8, 4/9, 5/10}
\qquad
\twolinematrix[b]{
  1/6, 2/7, 3/8, 4/9, 5/10,
  6/11, 7/12, 8/13, 9/14, 10/15
}
\]
\end{document}
```

It is recommended to transfer action to an expl3 command as soon as the code is more than a few lines, as we do here. The optional argument defaults to empty; this exploits the uniform syntax of the `matrix`-like environments of amsmath, so we can use `b` for brackets or `p` for parentheses and so on.

The main function clears two `seq` variables and proceeds to repopulate them. Since the argument is a comma-separated list, it's convenient to use `\clist_map_function:nN` that passes each item to the specified function, in this case `\twoline_add:n`, which will add the parts to the two sequences. The function separates the two parts by using an auxiliary function `\__twoline_add:w`, whose definition is clear: the first argument is whatever precedes the slash, the second argument is what's behind it. The "quark" `\q_stop` is customary here, the analog of `\@nil` in the LaTeX kernel.

After having populated the two sequences, we can deliver them to form the two rows of the matrix. Since `\seq_use:Nn` delivers its output in one swoop, there's no problem of already being in an alignment: typesetting will only start when the action of `\seq_use:Nn` has ended.

You might wonder why the functions are defined with the `protected` variant of `\cs_new`. It's because these functions perform assignments, in this case adding items to sequences, so they should not be expanded in a full expansion context. The unadorned `\cs_new:Nn` function should only be used for functions that can be fully expanded. This helps in avoiding the ever-loved *fragile commands*.

Another point is the declaration of function parameters: `\cs_new_protected:Nn` can deduce the parameter text from the function name being defined. On the other hand, `\cs_new_protected:Npn` needs the parameter text to be fully spelled out. In the case of `\__twoline_add:w` it is necessary because we use delimited arguments; the signature is thus `w`, for *weird*.

It would take too long to fully explain the double underscores; the idea is that functions or variables whose names start with a double underscore are *private*, whereas the others are *public*. For personal macros the distinction is a bit foggy, but it becomes important for package code: package writers are allowed to use the public functions of another package, but not the private ones, under the assumption that the syntax and action of the public functions are stable, whereas the private functions realizing the actual implementation may vary with time.

If the first row always has a sequence of integers in the natural order, we can simplify the input:

```
\NewDocumentCommand{\twolinematrix}{O{}m}
 {
  \twoline_matrix:nn { #1 } { #2 }
 }

\clist_new:N \l__twoline_row_clist

\cs_new_protected:Nn \twoline_matrix:nn
 {
  \clist_set:Nn \l__twoline_row_clist { #2 }
  \begin{#1matrix}
  1 % start at 1
  \use:x
   {
    \int_step_function:nnnN
     { 2 }
     { 1 }
     { \clist_count:N \l__twoline_row_clist }
     \__twoline_addindex:n
   }
  \\
  \clist_use:Nn \l__twoline_row_clist { & }
  \end{#1matrix}
 }
\cs_new:Nn \__twoline_addindex:n
 {
  & #1
 }
\ExplSyntaxOff
```

The calls would then be like

```
\twolinematrix{6, 7, 8, 9, 10}
\twolinematrix[b]{
  6, 7, 8, 9, 10,
  11, 12, 13, 14, 15
}
```

Here we exploit the fact we can access the number of items in a stored `clist` so we can easily generate the tokens `&2&3&...&`n by fully expanding with `\use:x` the function `\int_step_function:nnnN`. I leave filling in the details as an exercise to the reader. Time to move on.

3 String manipulation

LaTeX users sometimes have weird ideas like setting the title of a document based on the file name.[3] For instance, from a file name such as

`Chapter-Name_of_Section.tex`

the document should be titled

Chapter: Name of Section

[3] https://tex.stackexchange.com/questions/394489/

Not that I recommend such an approach, but at least it provides an occasion for describing some useful expl3 functions. We need to replace, in an expandable way, the hyphen with a colon plus space, and underscores with spaces.

expl3 defines `\c_sys_jobname_str` as its alias for the TEX primitive `\jobname`; we now make our acquaintance with another data type, namely `str` (string). The tokens in a `str` variable are stored 'literally'.[4] We also see here an example of a *constant*, that is, a variable whose value should never change.

Our approach is to examine the tokens in the file name one by one and output a replacement if needed:

```
\ExplSyntaxOn
\NewExpandableDocumentCommand{\massagedjobname}{}
 {
  \str_map_function:NN \c_sys_jobname_str
                       \ddas_jobname:n
 }
\cs_new:Nn \ddas_jobname:n
 {
  \str_case:nnF { #1 }
   {
    { - }{ :~ }
    { _ }{ ~ }
   }
   { #1 }
 }
\ExplSyntaxOff
```

Then one can do `\title{\massagedjobname}`. We pass each token to `\ddas_jobname:n`, which does the comparison: if the token is in the list in the second argument, then the corresponding replacement is done; in case of no match, the `F` branch is followed and, in this case the token itself is output. As all kernel function with `TF` branching, also `\str_case:nnTF` actually comes in four flavors

```
\str_case:nnTF
\str_case:nnT
\str_case:nnF
\str_case:nn
```

The true branch is followed when there is a match, but here we don't need it, so we can omit it using the third variant.

Again, there are classical TEX methods, but this has the big advantage of not requiring nested conditionals which would become very cumbersome when more than a couple of replacements are needed.

One might object that what we get doesn't have the correct category codes. Here's a different approach that also resets letters to category code 11.

```
\ExplSyntaxOn
\str_new:N \g_ddas_jobname_str
\NewDocumentCommand{\computetitle}{m}
 {
  \str_gset_eq:NN
   \g_ddas_jobname_str
   \c_sys_jobname_str
  \str_greplace_all:Nnn
   \g_ddas_jobname_str
   { - } { :~ }
  \str_greplace_all:Nnn
   \g_ddas_jobname_str
   { _ } { ~ }
  \tl_gset_rescan:Nnx
   #1
   { }
   { \str_use:N \g_ddas_jobname_str }
 }
\ExplSyntaxOff
\computetitle{\massagedjobname}
```

This is not expandable, but that is not a problem here, since what we need is a macro that holds the title. We perform the replacements in a more efficient fashion and then rescan the string so that the right category codes are assigned. This has to be done outside the scope of `\ExplSyntaxOn`, where the colon is a letter and the space is ignored. Global assignments are used, because the macro should be globally available; it wouldn't make a big difference here, but following a scheme is always best. The working of `\str_greplace_all:Nnn` should be clear from the function's name, and the syntax is as uniform as possible: `str` refers to the 'string' module, `greplace` stands for 'global replace'; the first argument is the `str` variable in which we want to do the replacement, the second argument is the search string, and the final one is the replacement string. There's a similar set of functions for `tl` variables. The second argument to `\tl_gset_rescan:Nnn` is for local assignments of category codes, but we need none.

Wait! Why is it `\tl_gset_rescan:Nnx`? This is a great feature of expl3. We can define *variants* of already existing functions. The argument type `x` means: a normal braced argument that is fully expanded before being passed to the main function. The expl3 kernel provides a definition for `\tl_gset_rescan:Nnn` and then has

```
\cs_generate_variant:Nn \tl_gset_rescan:Nnn { Nnx }
```

Thus, suppose we have a macro `\foo` that takes two arguments and we want to call it by first fully expanding the second argument. The classical approach would be:

```
\newcommand{\fooexpii}[2]{%
  \edef\@tempa{#2}%
  \expandafter\fooexpii@aux\expandafter{\@tempa}{#1}%
}
\newcommand{\fooexpii@aux}[2]{%
  \foo{#2}{#1}%
}
```

[4] For TEX hackers: as characters with category code 12, except for spaces that have their usual category code of 10.

Indeed, the variant defined above does essentially this, but the nice thing is that we don't need to know the details, just enjoy the result.

3.1 Colorizing capital letters

Another funny request is to change the color of capital letters in a given token list.[5] For this a different approach is needed, with regular expressions: expl3 has a powerful regular expression engine, tailored for the special quirks of TEX.

```
\documentclass{article}
\usepackage{xparse}
\usepackage{xcolor}

\ExplSyntaxOn
\NewDocumentCommand{\colorcap}{ O{blue} m }
 {
  \sheljohn_colorcap:nn { #1 } { #2 }
 }

\tl_new:N \l__sheljohn_colorcap_input_tl
\cs_new_protected:Npn \sheljohn_colorcap:nn #1 #2
 {
  % store the string in a variable
  \tl_set:Nn \l__sheljohn_colorcap_input_tl { #2 }
  \regex_replace_all:nnN
    % search a capital letter (or more)
    { ([A-Z]+ | \cC.\{?[A-Z]+\}?) }
    % replace the match with \textcolor{#1}{<match>}
    { \c{textcolor}\cB\{#1\cE\}\cB\{\1\cE\} }
    \l__sheljohn_colorcap_input_tl
  \tl_use:N \l__sheljohn_colorcap_input_tl
 }
\ExplSyntaxOff

\begin{document}
\colorcap{\'Once \r{U}pon a Time}

\colorcap[red]{Once Upon a Time}
\end{document}
```

We store the input in a `tl` (token list) variable and then proceed to search for capital letters with `[A-Z]+` (one or more) or sequences formed by

1. any control sequence, denoted by `\cC.`,
2. an optional open brace, `\{?`,
3. one or more capital letters, and
4. an optional close brace, `\}?`.

A match is replaced by `\textcolor{#1}{`⟨*match*⟩`}`. The syntax of the replacement text is admittedly peculiar, but it's necessary for getting the correct tokens with the likewise peculiar TEX properties. Our output is:

Ònce Ůpon a Time
Once Upon a Time

Variables of type `tl` are simply containers of TEX tokens.

While tokens in a `tl` variable are usually stored with their category code, we can rescan them. An example where this is useful is for splitting Windows-style paths, which can use the backslash instead of the slash of other operating systems.[6]

3.2 Menu sequences

We'd like to be able to say `\menu{1,2,3,4}` and treat specially the first and last item, with provision for a single item. The macro should be able to specify a different separator, for cases such as

```
\menu[/]{C:/A/B/C}
\menu*{C:\A\B\C}
```

The *-variant will use the backslash as separator. The code is rather longish, but instructive.

```
\documentclass{article}
\usepackage{xparse}

\ExplSyntaxOn

% user level commands
\NewDocumentCommand{\setmenuseparator}{ m }
 {
  \tobi_menu_setsep:n { #1 }
 }
\NewDocumentCommand{\menu}{ s o m }
 {
  \group_begin:
  \IfValueT{#2}{ \tobi_menu_setsep:n { #2 } }
  \IfBooleanTF{#1}
   {
    \tobi_menu_process_rescan:n { #3 }
   }
   {
    \tobi_menu_process:n { #3 }
   }
  \group_end:
 }

% variables
\seq_new:N \l_tobi_menu_seq
\tl_new:N \l_tobi_menu_sep_tl
\tl_set:Nn \l_tobi_menu_sep_tl { , } % default
\tl_new:N \l_tobi_first_tl
\tl_new:N \l_tobi_last_tl
\tl_new:N \l_tobi_input_tl

% internal functions
\cs_new:Nn \tobi_menu_setsep:n
 {
  \tl_set:Nn \l_tobi_menu_sep_tl { #1 }
 }

\cs_new:Npn \tobi_menu_process:n #1
 {
  \seq_set_split:NVn \l_tobi_menu_seq
   \l_tobi_menu_sep_tl
   { #1 }
  \tobi_premenu:
  \int_case:nnF { \seq_count:N \l_tobi_menu_seq }
   {
    { 0 } { EMPTY }
```

[5] https://tex.stackexchange.com/questions/173209/

[6] https://tex.stackexchange.com/questions/44961/

```
    { 1 } { \tobi_singlemenu:n { #1 } }
   }
   {
    \seq_pop_left:NN \l_tobi_menu_seq \l_tobi_first_tl
    \seq_pop_right:NN \l_tobi_menu_seq \l_tobi_last_tl
    \tobi_firstmenu:V \l_tobi_first_tl
    \seq_map_function:NN
     \l_tobi_menu_seq
     \tobi_midmenu:n
    \tobi_lastmenu:V \l_tobi_last_tl
   }
  \tobi_postmenu:
 }

\cs_new_protected:Npn \tobi_menu_process_rescan:n #1
 {
  \group_begin:
  \tl_set_eq:NN \l_tobi_menu_sep_tl \c_backslash_str
  \tl_set_rescan:Nnn \l_tobi_input_tl
   { \char_set_catcode_other:N \\ }
   { #1 }
  \tobi_menu_process:V \l_tobi_input_tl
  \group_end:
 }
\cs_generate_variant:Nn \seq_set_split:Nnn { NV }
\cs_generate_variant:Nn \tobi_menu_process:n { V }

% customize to suit
\cs_new_protected:Nn \tobi_premenu:
 { \fbox{\strut pre} }
\cs_new_protected:Nn \tobi_postmenu:
 { \fbox{\strut post} }
\cs_new_protected:Nn \tobi_firstmenu:n
 { \fbox{\strut #1~(first)} }
\cs_generate_variant:Nn \tobi_firstmenu:n { V }
\cs_new_protected:Nn \tobi_midmenu:n
 { \fbox{\strut #1~(mid)} }
\cs_new_protected:Nn \tobi_lastmenu:n
 { \fbox{\strut #1~(last)} }
\cs_generate_variant:Nn \tobi_lastmenu:n { V }
\cs_new_protected:Nn \tobi_singlemenu:n
 { \fbox{\strut #1~(single)} }

\ExplSyntaxOff

\begin{document}

\menu{1,2,3,4}\par\medskip
\menu{Single Element}\par\medskip
\menu{A,B,C,D,E}\par\medskip
\menu[/]{A/B/C/D/E}\par\medskip

\setmenuseparator{/}
\menu{C:/A/B/C}\par\medskip

\menu*{C:\A\B\C}

\end{document}
```

We start off with an interesting application of input splitting; we can set a `seq` variable to the items we obtain from breaking the input at the specified sequence of tokens. The function for this is `\seq_set_split:Nnn`, whose second argument is the chosen separator. However, in this application the separator is variable, so we define a variant `\seq_set_split:NVn`. The `V` argument type means "brace the contents of the specified variable and pass it as if it were a normal argument". In classical terms the action is similar to

```
\expandafter\foo\expandafter\xyz
\expandafter{\baz}{arg}
```

where `\foo` is the three-argument macro and `\baz` is a container.

We then branch according to the number of items in the sequence. When we have more than one item, we separate off the first and the last to receive special treatment: with `\seq_pop_left:NN` we remove the leftmost item from the `seq` and store it in a `tl` variable. Then we process the first item, the middle items, and the last. Again, defining a variant is handy for processing the special items: we define a function for the explicit argument and then a `V` variant thereof.

For the *-variant, the input is first rescanned, making the backslash a printable character, and the separator is set to `\` using `\c_backslash_str`, a predefined string. Then `\tl_menu_process:V` is used so as to 'recycle' the standard function without having to bother with `\expandafter`.

Everything is done in a group in order to allow local setting of the separator, which can also be set (conforming to the standard scoping rules) by `\setmenuseparator`. Here's the output:

pre	1 (first)	2 (mid)	3 (mid)	4 (last)	post

pre	Single Element (single)	post

pre	A (first)	B (mid)	C (mid)	D (mid)	E (last)	post

pre	A (first)	B (mid)	C (mid)	D (mid)	E (last)	post

pre	C: (first)	A (mid)	B (mid)	C (last)	post

pre	C: (first)	A (mid)	B (mid)	C (last)	post

3.3 Doubling backslashes in auxiliary file

Another example of input manipulation is the request for writing to an auxiliary file, but doubling all backslashes, for feeding to an external program.[7]

```
\documentclass{article}
\usepackage{xparse}

\ExplSyntaxOn
\NewDocumentCommand{\setupstream}{ O{default} m }
 {
  \iow_new:c { g_mblanc_dbswrite_#1_iow }
  \iow_open:cn { g_mblanc_dbswrite_#1_iow } { #2 }
  \AtEndDocument
   {
```

[7] https://tex.stackexchange.com/questions/402011/

```
    \iow_close:c { g_mblanc_dbswrite_#1_iow }
   }
 }

\NewDocumentCommand{\dbswrite}{ s O{default} m }
 {
  \IfBooleanTF { #1 }
   {% argument is a macro
    \mblanc_dbswrite:nV { #2 } #3
   }
   {% argument is explicit
    \mblanc_dbswrite:nn { #2 } { #3 }
   }
 }

\tl_new:N \l__mblanc_dbswrite_text_tl

\cs_new_protected:Nn \mblanc_dbswrite:nn
 {
  \tl_set:Nx
   \l__mblanc_dbswrite_text_tl
   { \tl_to_str:n { #2 } }
  \tl_replace_all:Nxx \l__mblanc_dbswrite_text_tl
   { \c_backslash_str }
   { \c_backslash_str \c_backslash_str }
  \iow_now:cV
   { g_mblanc_dbswrite_#1_iow }
   \l__mblanc_dbswrite_text_tl
 }
\cs_generate_variant:Nn \mblanc_dbswrite:nn { nV }

\cs_generate_variant:Nn \tl_replace_all:Nnn { Nxx }
\cs_generate_variant:Nn \iow_now:Nn { cV }

\ExplSyntaxOff

\setupstream{\jobname.TESTFILE}
\setupstream[secondary]{\jobname.TESTFILESEC}

\newcommand{\test}{%
  Here are my contents: \UndefinedMacro and \\%
}

\begin{document}

\dbswrite{%
  Here are my contents:
  \UndefinedMacro and \\
}
\dbswrite[secondary]{%
  Here are my contents:
  \UndefinedMacro and \\%
}

\dbswrite*{\test}
\dbswrite*[secondary]{\test}

\end{document}
```

We find here still another data type, `iow` (input/output write). This is a good place to discuss a nice feature of expl3 regarding input and output streams. Since conventional TEX engines have a very limited number of streams (16), a new stream is allocated from a pool and when the stream is closed that stream is available again, in contrast to what happens in current LATEX (and plain TEX). Here this is irrelevant, as the stream is only closed at the end of the document, but it can be useful in other applications.

With `\iow_new:N` we can allocate a new write stream, but here we may need more than one, with a symbolic name. Thus we use a variant with the `c` type, so that the braced argument is turned into a control sequence, the counterpart of the classical

```
\expandafter\foo\csname baz\endcsname
```

This way, we can easily use a variable name. The optional argument defaults to `default`, but we can set up as many as we want. Thus the macro `\dbswrite` takes an optional argument for the symbolic name of the stream, and also has a *-variant for the case when we want to pass a `tl` argument (here a classical parameterless macro).

The argument is first so-called "stringified" with `\tl_to_str:n`, then backslashes are doubled with `\tl_replace_all:Nxx`, and finally, the contents are written out. The expl3 kernel doesn't provide every possible variant, so we need to do

```
\cs_generate_variant:Nn \tl_replace_all:Nnn {Nxx}
\cs_generate_variant:Nn \iow_now:Nn {cV}
```

It's no problem if some other code, maybe from a package we load, does the same, because an already existing variant will cause the code above to do nothing and the variants are defined in a uniform way. Similarly we need `\mblanc_dbswrite:nV` for the *-variant. Here we can see why we want `\NewDocumentCommand` to generally do only "argument parsing and normalization" and then pass control to an internal public function: we just need to concentrate on `\mblanc_dbswrite:nn` and a variant will cope with the other case.

The generated files will be identical and contain

```
Here are my contents: \\UndefinedMacro and \\\\
Here are my contents: \\UndefinedMacro and \\\\
```

4 Macro factory

In several cases one has to build several macros following a certain scheme. The mapping facilities of expl3 help in writing compact code.

The first example is about defining macros that expand to the items in a given list.[8] To begin, from `\DefinitionVariables{abc,def}` we'd like to define `\variableI` and `\variableII` expanding to `abc` and `def` respectively. Here's the code:

```
\NewDocumentCommand{\DefinitionVariables}{m}
 {
  \int_zero:N \l_tmpa_int
  \clist_map_inline:nn { #1 }
   {
```

[8] https://tex.stackexchange.com/questions/367335/

```
    \int_incr:N \l_tmpa_int
    \tl_clear_new:c
     {
      variable \int_to_Roman:n { \l_tmpa_int }
     }
    \tl_set:cn
     {
      variable \int_to_Roman:n { \l_tmpa_int }
     }
     { ##1 }
   }
 }
```

Actually, we're slightly abusing the language for defining a 'user level' macro with (a variant of) `\tl_set:Nn`.

The given list is mapped by passing each item to the second argument, where the current item is referred to as `#1`; here the hash mark needs to be doubled because we're in the body of a definition. Compare this with the standard `\@for` cycle, where the current item is stored in a macro, which typically needs to be expanded, often in an awkward way.

We can avoid allocating a new `int` (integer) variable and use the scratch one provided by the kernel. An alternative way could be

```
\NewDocumentCommand{\DefinitionVariables}{m}
 {
  \int_step_inline:nnnn
   { 1 } % start
   { 1 } % step
   { \clist_count:n { #1 } } % end
   {
    \tl_clear_new:c
     {
      variable \int_to_Roman:n { ##1 }
     }
    \tl_set:cx
     {
      variable \int_to_Roman:n { ##1 }
     }
     { \clist_item:nn { #1 } { ##1 } }
   }
 }
```

but this is less efficient because the `clist` needs to be scanned at each step. However, this is a nice way to show how we can do integer-based cycles.

4.1 Symbol abbreviation macro sets

A possibly better example of a macro factory is the following: we want to define `\CC` to stand for `\mathbb{C}` and also `\cS` to stand for `\mathcal{S}`. Of course we'd like to add other similar symbols with as little burden as possible.[9]

```
\ExplSyntaxOn
\NewDocumentCommand{\makeabbrev}{mmm}
 {
  \yoruk_makeabbrev:nnn { #1 } { #2 } { #3 }
 }
\cs_new_protected:Nn \yoruk_makeabbrev:nnn
 {
  \clist_map_inline:nn { #3 }
   {
    \cs_new_protected:cpn { #2 } { #1 { ##1 } }
   }
 }
\ExplSyntaxOff

\makeabbrev{\mathbb}{#1#1}{C,N,Q,Z,D,R,T}
\makeabbrev{\mathcal}{c#1}{A,B,C,S}
```

Tricky code, isn't it? If we try it, we'll find that `\CC` indeed expands to `\mathbb{C}`. The trick is that when `#2` is picked up, the hash marks are doubled (this is a general TEX feature). When performing each cycle in the first call of `\makeabbrev`, what TEX sees is

```
\cs_new_protected:cpn { #1#1 } { \mathbb { #1 } }
```

(because double hash marks are reduced to single during macro expansion) and, when the current item is `C`, this becomes

```
\cs_new_protected:cpn { CC } { \mathbb{C} }
```

which is exactly what we need. Here we must use the `:cpn` signature for `\cs_new_protected` because the macro we're defining has no signature itself, so the (empty) parameter text is mandatory, as it cannot be deduced. As before, `c` stands for 'make a control sequence out of the argument'.

What if we wanted to define `\AA` to `\ZZ` in one fell swoop and maybe also `\abf` to `\zbf` to stand for `\mathbf{a}` and so on? We can use the command `\int_step_inline:nnnn` to populate a `clist` and then do the same; a check whether we're redefining an existing control sequence is added as otherwise we'd get errors for `\AA` and `\SS`.

```
\ExplSyntaxOn
\NewDocumentCommand{\makeabbreviations}{mmmm}
 {% #1 = wrapper macro
  % #2 = template
  % #3 = starting letter
  % #4 = ending letter
  \clist_clear:N \l_tmpa_clist
  \int_step_inline:nnnn
   { `#3 } % start
   { 1 }   % step
   { `#4 } % end
   {% populate a clist
    \clist_put_right:Nx
     \l_tmpa_clist
     { \char_generate:nn { ##1 } { 12 } }
   }
  \clist_map_inline:Nn \l_tmpa_clist
   {
    \cs_if_exist:cTF { #2 }
     {
      \msg_term:n
       {
        Not~redefining~\c_backslash_str#2
       }
     }
     {
```

[9] https://tex.stackexchange.com/questions/207985/

```
      \cs_new_protected:cpn { #2 } { #1 { ##1 } }
     }
   }
 }
\ExplSyntaxOff

\makeabbreviations{\mathbb}{#1#1}{A}{Z}
\makeabbreviations{\mathbf}{#1bf}{a}{z}
```

In the log file and on the console we'd see

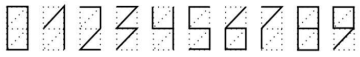
```
************************************************
* Not redefining \AA
************************************************
************************************************
* Not redefining \SS
************************************************
```

5 Graphics

I'd like to end this showcase with some new features of expl3 regarding graphic inclusion. The team (primarily Joseph Wright) is currently working on a set of APIs for the graphics driver meant to implement the same APIs as PGF, with different names, of course.

Some basic calls are already provided by the current kernel (release of 21 February, 2018, at this writing).

A funny question about printing numbers in the style required by the Soviet Union postal service appeared in October 2017.[10] These numerals look like this:



The idea is to use another expl3 data type, namely `prop` (property list). A property list is a container where data is identified by a key, in this case the digit. We can extract an item, typically consisting of code, by using its key.

So we start with

```
\prop_new:N \g_torcli_sovietdigits_prop

\prop_gput:Nnn \g_torcli_sovietdigits_prop { 0 }
 {
   ⟨code for 0⟩
 }
```

and so on for the other digits. Then we define an interface

```
\NewDocumentCommand{\postalcode}{O{}m}
 {
  \mbox
   {
    \keys_set:nn { torcli/sovietdigits } { #1 }
    \torcli_sovietdigits_print:n { #2 }
   }
 }
```

and now it's down to using the code for the various digits stored in the `prop`. I'll not go into the details of the key-value interface, suffice it to say that the code is defined in terms of some parameters, using the new LaTeX3 graphics commands to draw the necessary lines.

In the fully expl3 version, the code for 0 and 6 is

```
\prop_gput:Nnn \g_sovietdigits_prop { 0 }
 {
  \__sovietdigits_moveto:nn {0}{0}
  \__sovietdigits_lineto:nn {1}{0}
  \__sovietdigits_lineto:nn {1}{2}
  \__sovietdigits_lineto:nn {0}{2}
  \driver_draw_closestroke:
 }

\prop_gput:Nnn \g_sovietdigits_prop { 6 }
 {
  \__sovietdigits_moveto:nn {1}{2}
  \__sovietdigits_lineto:nn {0}{1}
  \__sovietdigits_lineto:nn {0}{0}
  \__sovietdigits_lineto:nn {1}{0}
  \__sovietdigits_lineto:nn {1}{1}
  \__sovietdigits_lineto:nn {0}{1}
  \driver_draw_stroke:
 }
```

where the functions `\__sovietdigits_moveto:nn` and `\__sovietdigits_lineto:nn` are simply syntactic sugar around the basic calls:

```
% Syntactic sugar
\cs_new_protected:Nn \__sovietdigits_moveto:nn
 {
  \driver_draw_moveto:nn
   { #1 \l_sovietdigits_width_dim }
   { #2 \l_sovietdigits_width_dim }
 }
\cs_new_protected:Nn \__sovietdigits_lineto:nn
 {
  \driver_draw_lineto:nn
   { #1 \l_sovietdigits_width_dim }
   { #2 \l_sovietdigits_width_dim }
 }
```

Happy LaTeX3ing!

◇ Enrico Gregorio
Dipartimento di Informatica
Università di Verona
and
LaTeX Team
enrico.gregorio@univr.it

[10] https://tex.stackexchange.com/questions/394616/

Three-dimensional graphics with TikZ/PSTricks and the help of GeoGebra

Luciano Battaia

Abstract

In this article we consider the opportunity of using dynamic geometry software, such as GeoGebra, to allow easy exporting of three-dimensional geometric pictures, with subsequent 2D parallel projection, in PGF/TikZ or PSTricks code. The help of software like GeoGebra considerably simplifies the production of very complex pictures in LaTeX code, requiring only a basic knowledge of PGF/TikZ or PSTricks languages and taking advantage of a substantially mouse driven program. All examples and sample code here are in PGF/TikZ, but almost nothing changes if one prefers PSTricks.

1 Introduction

LaTeX users, particularly those writing scientific papers, have always had a need for high-quality vector graphics, including labels, that fit the style of the rest of their documents.

There is no special problem in the case of two-dimensional graphics, and the two most widespread tools PSTricks and PGF/TikZ (that from now on will only be mentioned merely as TikZ), together with their derived packages, solve almost every problem very well. As Claudio Beccari has shown [2], LaTeX's basic *picture* environment is sufficient for many situations.

Things change substantially if we are interested in three-dimensional graphics. Plots of two-variable functions and of various kinds of surfaces can easily be handled using dedicated packages, for instance pst3dplot or pgfplots. Also, for geometric figures some very interesting packages are available, for example pst-solides3d or pst-3d in the PSTricks family or tikz-3dplot in the TikZ family, but in all cases a rather deep knowledge of programming techniques in PSTricks or TikZ is needed and, in our opinion, this is not at all easy for the average user.

External programs that produce PSTricks or TikZ codes can help, for instance *Sketch* by Eugene Ressler (see for example [3]) and *TEXgraph* by Patrick Fradin (`texgraph.tuxfamily.org/`). The last one in particular is very powerful and can also produce *POV-Ray* code, but, again, it is not within the reach of most users. Almost the same remarks apply to *Asymptote*, whose code can be directly included in a LaTeX source through the asymptote package.

An interesting and detailed introduction to the problem of producing three-dimensional graphics with TikZ can be found in an article by Keith Wolcott [5]. It was in fact the reading of this article that led us to study the problem in order to find a more accessible solution. Wolcott's article ends with a figure which shows only the partial solution of what had been the main purpose of the project: the drawing of two spheres and their circle of intersection. The author himself points out that the figure needs more work.

This is the reason why we begin this article with figure 1, which exactly reaches Wolcott's goal. Explanations on how we obtain it will be given later, but we immediately point out that our approach to the problem is completely different from Wolcott's.

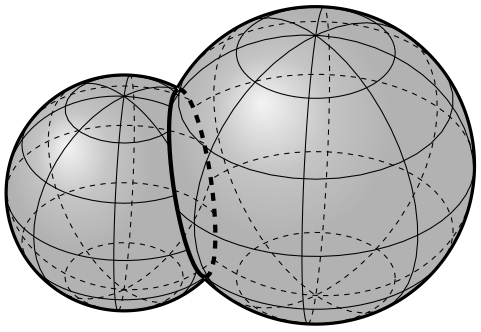


Figure 1: The intersection of two spheres with the circle of intersection

For the sake of completeness, we mention that a slightly different version of this article, in Italian, can be found in [1].

2 The coming of GeoGebra on the scene

For educational reasons we have been using *GeoGebra* for a long time, both because its non-commercial use is free and because its basic use is extremely simple. With reference to the problem we are dealing with, the interesting thing is the possibility of producing complex two-dimensional figures and exporting them in PSTricks or TikZ code, that can then be copied and pasted directly into a LaTeX source with only very limited adaptations, mainly regarding correct label positioning. The required knowledge of LaTeX packages is minimal and manageable for even inexperienced users. In short, anyone can produce even complex figures to be included in LaTeX documents with a WYSIWYG technique and extensively using the mouse. This seems far from what a LaTeX user normally does, but we think that in the case of graphics this strategy is preferable for many users. Of course one must know GeoGebra well enough, but this does not require the study of long and complex handbooks and, at any rate, dynamic geometry

software is of great help in experimenting with the construction of technical figures. An example of a complex 2D figure easily produced in GeoGebra and exported into Ti*k*Z almost without intervention in the generated code is shown in figure 2.

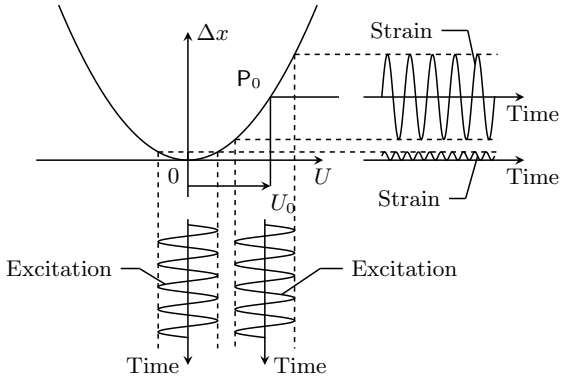


Figure 2: A picture produced with GeoGebra and exported into Ti*k*Z, used for a master's degree thesis

GeoGebra can export figures into both PSTricks and Ti*k*Z code (and even into *Asymptote*); in this article we consider only the case of Ti*k*Z, with which all the figures shown are realized. However, as mentioned above, substantially nothing changes if you prefer PSTricks, because, apart from some adaptations and some limited work to clean up the code, everything is automatically produced. For this reason also, only a few fragments of source code will be included. In addition, it should be noted that the generated code is not very interesting, as it consists almost exclusively of `\draw` instructions; all needed calculations have already been done by GeoGebra.

Some time ago a new version of GeoGebra (Geogebra Classic 5.0) which supports three-dimensional graphics was released. Unfortunately, for this 3D version no export into a LaTeX format has yet been implemented and, in our opinion, this will not be possible, at least not in a reasonably short time. Because of this limitation we decided to experiment with the possibility of directly executing a 3D to 2D projection in GeoGebra and then exporting it into Ti*k*Z code. Indeed, each 3D figure is just an appropriate 2D projection of a three-dimensional object.

Keeping this in mind, the first thing we tried to reproduce is a sphere originally drawn by Tomasz M. Trzeciak [4]; it was also reproduced by Keith Wolcott in [5]. Please compare Trzeciak's original (figure 3) with ours (figure 4).

The two pictures are almost identical but the Ti*k*Z codes are indeed completely different; you can compare them in detail in the Italian version of this article [1]. Here we only want to point out the fact

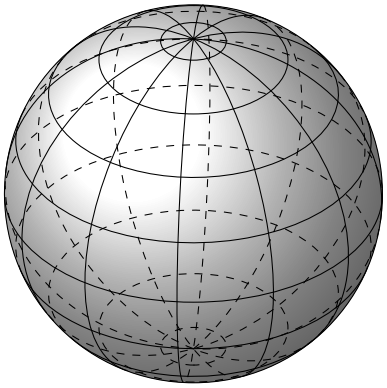


Figure 3: Sphere with meridians and parallels, produced by Tomasz Trzeciak using PGF/Ti*k*Z

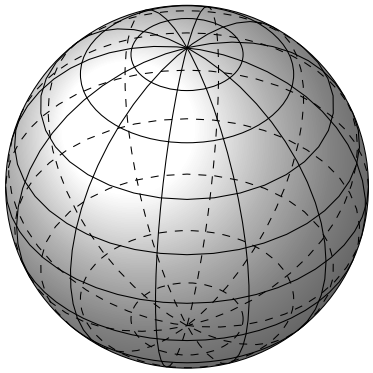


Figure 4: Sphere with meridians and parallels, produced with code exported from GeoGebra

that Trzeciak's code is much more concise and elegant, but it requires a deep knowledge of PGF programming. In fact you must first instruct PGF to make the correct calculations for the visible and invisible parts of each latitude or longitude circle, using appropriate PGF macros, and only after that you can draw the circles. In our code all calculations are made by GeoGebra, and only the drawing part is left to Ti*k*Z.

3 Some maths behind the scene

GeoGebra is a very well structured and powerful program for dynamic geometry. There are two different windows for 2D graphics, a window for 3D graphics, a fairly complete spreadsheet, a probability calculator and an algebra window where you can read the coordinates of the points, the equations of the curves, and so on. The very important feature is that all the windows can interact with each other. Regarding our problem, all that is obtained in the 3D window can be appropriately transferred to the main 2D window

(and then exported into LaTeX code). Moreover, it is interesting to note that GeoGebra is in any case "LaTeX oriented"; all textual annotations are inserted in the windows with LaTeX code.

Let us consider a Cartesian orthogonal system in three-dimensional space, that in GeoGebra is displayed in the 3D window, with an upward vertical z-axis; call α a rotation around the vertical axis and β a rotation around a horizontal axis. The parallel projection of this Cartesian system in a plane (that in our case will be the main 2D window of GeoGebra), can be obtained, for instance, with the following formulas:

$$\begin{aligned}\vec{i} &= \big(-\cos(\alpha), -\sin(\alpha)\sin(\beta)\big)\\ \vec{j} &= \big(\sin(\alpha), -\cos(\alpha)\sin(\beta)\big)\\ \vec{k} &= \big(0, \cos(\beta)\big)\end{aligned} \quad ,$$

where $\vec{i}$, $\vec{j}$, $\vec{k}$ are the vectors of the basis. If you set the origin to the point $\mathsf{O} = (0, 0)$, which is preferable, you must create two angle sliders with the names α and β; afterwards the basis vectors can be constructed with the following GeoGebra code:

$$\begin{aligned}\mathrm{i} &= \mathrm{Vector}[\mathsf{O}, (-\cos(\alpha), -\sin(\alpha)\sin(\beta))]\\ \mathrm{j} &= \mathrm{Vector}[\mathsf{O}, (\sin(\alpha), -\cos(\alpha)\sin(\beta))]\\ \mathrm{k} &= \mathrm{Vector}[\mathsf{O}, (0, \cos(\beta))]\end{aligned} \quad .$$

Now, if you consider a point $\mathsf{P} = (x_\mathsf{P}, y_\mathsf{P}, z_\mathsf{P})$ in the 3D window, its projection will be

$$\mathsf{P}' = x_\mathsf{P}\,\vec{i} + y_\mathsf{P}\,\vec{j} + z_\mathsf{P}\,\vec{k},$$

or, in the language of GeoGebra,

$$\mathsf{P}' = x(\mathsf{P})\,\mathrm{i} + y(\mathsf{P})\,\mathrm{j} + z(\mathsf{P})\,\mathrm{k}.$$

If you consider instead a curve C with parametric equations $(f(t), g(t), h(t))$, with the parameter t appropriately included between two extremes, its 2D projection, always in the GeoGebra language, will be

$$\begin{aligned}x' &= f(t)\,x(\mathrm{i}) + g(t)\,x(\mathrm{j}) + h(t)\,x(\mathrm{k})\\ y' &= f(t)\,y(\mathrm{i}) + g(t)\,y(\mathrm{j}) + h(t)\,y(\mathrm{k})\end{aligned} \quad .$$

These formulas allow the 2D projection of every figure made in the 3D window of GeoGebra. After that you can experiment to find the best view for the figure by changing the angles α and β, working in the 2D window; this is an important feature because in general it is very difficult to find the appropriate viewing angle, and only trying over and over again can lead to the solution. Naturally not even GeoGebra minimizes the problem of 3D graphics as it is clear that those who need images of this type must have a good mathematical preparation. Nothing is obtained for free!

In light of these formulas let's see in detail, as an example, how the sphere of figure 4 can be obtained. Begin by plotting the 3D sphere with center the origin and radius r and its 2D projection that is simply the circle with center the origin and again radius r. Next draw the parallels and meridians simply intersecting the sphere with appropriate planes. If, for instance, you need five parallels they will be found at the latitudes $-60°$, $-30°$, $0°$, $30°$, $60°$ and the corresponding planes have the following equations

$$\begin{aligned}z &= r\sin(-60°)\ ;\ z = -r\sqrt{3}/2\\ z &= r\sin(-30°)\ ;\ z = -r/2\\ z &= r\sin(0°)\ \ ;\ z = 0\\ z &= r\sin(30°)\ \ ;\ z = r/2\\ z &= r\sin(60°)\ \ ;\ z = r\sqrt{3}/2\end{aligned} \quad .$$

These planes can be plotted simply by writing the equations in the input bar. Now ask GeoGebra to find the intersection circle of the planes with the sphere and choose (for instance using the mouse) five points on each circle. After projecting these points on the 2D window, plot the conic through them, using the specific Command; this will be the projected parallel. Do the same for the meridians. Now, after choosing the best viewing angle, highlight the visible and invisible part of each ellipse. For the invisible part you can decide if you want to show it or not, you can choose a broken line, a reduced thickness, and so on. When everything is perfectly configured, export into Ti*k*Z (or PSTricks) and insert the code in your LaTeX document; it usually works very well and only small adaptations are normally needed, for instance regarding the position of the labels or if you need special shading. The technique that we have illustrated is absolutely basic; with a little experience in the use of GeoGebra, everything can be faster and further automated.

A point that deserves further attention from what has been previously described is how to treat the visible or invisible parts of the projected figure in GeoGebra. The 3D window of the software can automatically handle the visible or invisible parts, as shown in the screen shot of figure 5.

The projection of this picture on the 2D window produces an image where visible and invisible parts are plotted in the same style, as shown in figure 6; such a figure can't be exported as it is.

Now, by comparing the side by side images of the 3D and 2D windows, and using the Intersect command of Geogebra, one can correctly highlight the visible and invisible parts of each curve and finally obtain the image ready to export. It is shown in figure 7.

Regarding the intersection of two spheres, plotted in figure 1, there are no further complications since the intersection circle can be found directly by

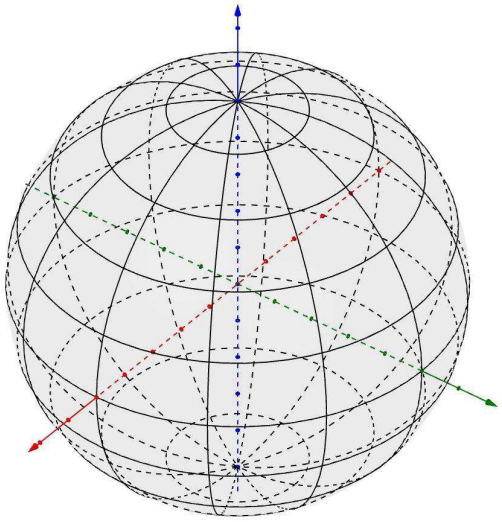


Figure 5: Screenshot of the main 3D window of GeoGebra for the production of the sphere of figure 4

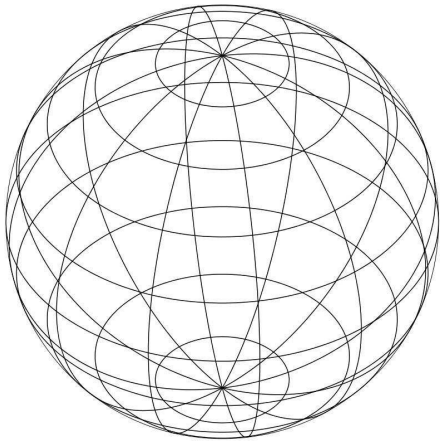


Figure 6: Screenshot of the 2D projection in Geogebra of the 3D window shown in figure 5

GeoGebra and then projected on the 2D window as described. In this case we have chosen not to show the overlapping parts of the spheres at all, in order to obtain a more readable figure.

4 A spiral on a sphere

The following example requires a minimum of extra mathematics, but no further work on the code. The goal is to plot a complex spiral, with endless turns, on a sphere, highlighting the property that the angle between the meridians and the spiral remains constant. The best way to solve the problem is to use

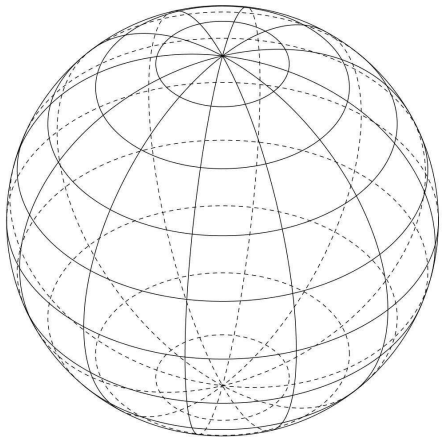


Figure 7: Screenshot of the main 2D window of Geogebra ready to be exported for the production of the sphere of figure 4

the following parametric equations of the spiral

$$\begin{aligned} x(t) &= \frac{r\cos t}{\sqrt{1+a^2t^2}} \\ y(t) &= \frac{r\sin t}{\sqrt{1+a^2t^2}}\,, \\ z(t) &= \frac{-art}{\sqrt{1+a^2t^2}} \end{aligned}$$

where r is the radius of the sphere and a is a parameter. It is preferable to set up r and a with sliders in GeoGebra and then choose the best values after testing different ones. The tracing of the tangent vectors and of the angles identified by them is straightforward.

The only thing that needs special attention is the fact that plots of lines such as the one needed here can't be drawn directly by Ti*k*Z and you need external software, for instance GNUPLOT, but this can be done in a straightforward way, and, in any case, GeoGebra automatically handles this problem in the export procedure! It should be noticed that PSTricks handles directly these situations. The final plot is shown in figure 8.

5 Polyhedra

One of the situations where GeoGebra's intervention is truly providential is the drawing of polyhedra and their developments; there are special routines to draw, in particular, Platonic solids and to show dynamically their development. The 2D projection of such figures is indeed very simple because you need only to find the correct position of the projected vertices, whose three-dimensional coordinates are automatically found by GeoGebra. Figure 9 shows the dodecahedron, while figure 10 shows a step towards its development in a plane.

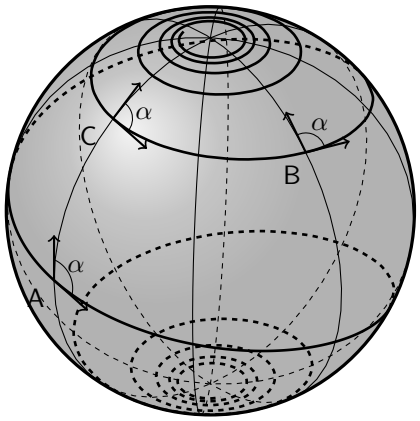


Figure 8: A spiral on a sphere

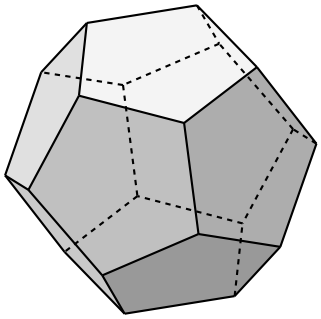


Figure 9: The regular dodecahedron

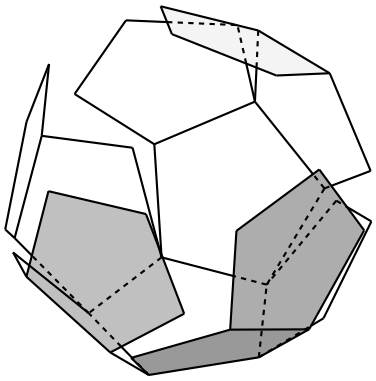


Figure 10: The regular dodecahedron: a step towards its development in a plane

Don't be fooled by the apparent simplicity of these pictures. The hand calculation of the coordinates of the vertices of the dodecahedron is not easy at all, and, even worse, their position during development!

Also, the drawing of the inscribed and circumscribed spheres is straightforward and you can see an example concerning the octahedron in figure 11.

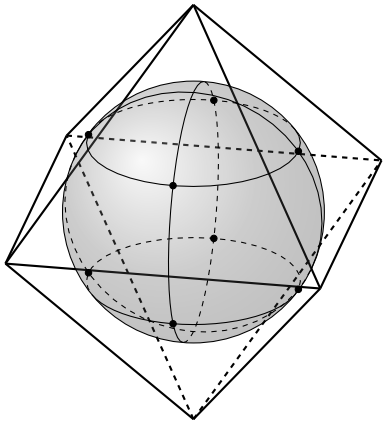


Figure 11: The regular octahedron and its inscribed sphere. The meridians and the parallels through four of the eight tangent points are highlighted

Even more important is the fact that the drawing of the curves described by the vertices during development is relatively straightforward. In GeoGebra every vertex can leave a track during the development and it is possible to project this track in the 2D window; it is now very simple to plot, using a GeoGebra macro, a Bezier curve, maybe at intervals, that approximates this track. Exporting this Bezier curve is a standard procedure. You can see an example in figure 12; the curve Γ is a complex curve, while all the others are simply circle arcs. It is in principle possible to find the parametric equations of Γ, but the use of GeoGebra capabilities makes everything extremely simple, without any calculation.

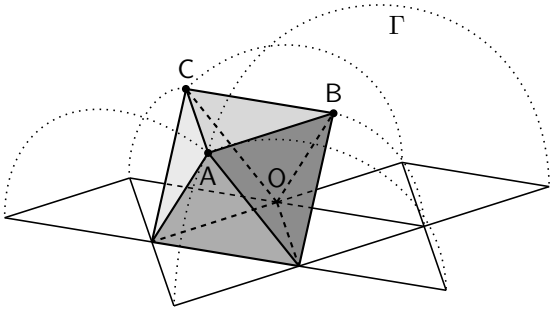


Figure 12: One of the different plane developments of the regular octahedron. The curves described by the vertices during development are highlighted

Once you have built the outline of a dodecahedron in the 3D window of GeoGebra, you can also experiment with interesting derived figures. An example is given in figure 13, where Leonardo's *Dodecahedron Planum Vacuum* is represented. Once more the calculation of the position of the vertices

of this figure is almost straightforward in GeoGebra, but it could be very difficult otherwise.

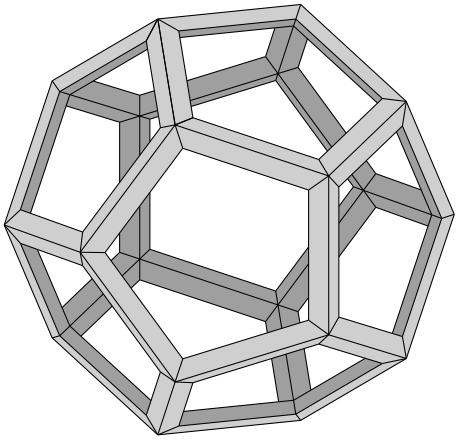


Figure 13: Dodecahedron Planum Vacuum, in Leonardo's style

6 The football

Once you have acquired familiarity with the Platonic solids, you can experience the expansion of the technique to other solids, i.e., the Archimedean solids. These can be obtained in various ways from the Platonics, for example by truncation starting from the vertices. In figure 14 we show the case of the icosahedron; given the Platonic solid, we consider, for each vertex, a sphere centered at the vertex itself and with variable radius. The intersection of this sphere with the sides of the polyhedron gives rise to regular pentagons and hexagons. The latter become regular when the radius of the sphere is exactly 1/3 of the side of the polyhedron and this situation corresponds to the truncated icosahedron. Using GeoGebra it is very easy again to document this process; simply project the truncation at the desired stage and then export it.

Figure 15 shows the final result. Nothing new is required in GeoGebra to obtain this last figure. It is exactly the same construction used for the previous figure 14, only with a different radius for the truncating spheres.

As is well known, the football is simply the projection of the truncated icosahedron on the circumscribed sphere. This can be achieved in different ways. In our opinion the simplest one is to project each side of the polyhedron onto the sphere by means of a parametric equation and then again to project the obtained arc in the 2D window. Following we describe the outline of this technique. Given a segment $\overline{\mathsf{AB}}$ with bounds $(x_\mathsf{A}, y_\mathsf{A}, z_\mathsf{A})$ and $(x_\mathsf{B}, y_\mathsf{B}, z_\mathsf{B})$, write the standard parametric equations of the segment

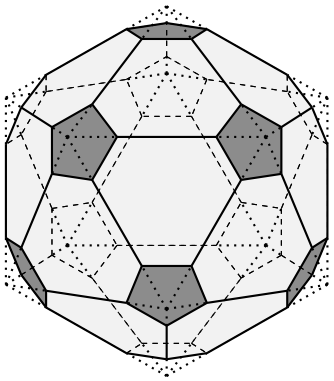


Figure 14: Outline of the truncation of the icosahedron starting from the vertices

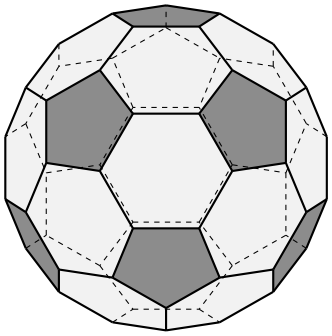


Figure 15: The truncated icosahedron

itself:

$$\mathsf{P}(t)\colon \begin{cases} f(t) = x_\mathsf{A} + (x_\mathsf{B} - x_\mathsf{A})t \\ g(t) = y_\mathsf{A} + (y_\mathsf{B} - y_\mathsf{A})t \\ h(t) = z_\mathsf{A} + (z_\mathsf{B} - z_\mathsf{A})t \end{cases}, \ 0 \le t \le 1.$$

Then find the norm of $\mathsf{P}(t)$:

$$||\mathsf{P}(t)|| = \sqrt{f^2(t) + g^2(t) + h^2(t)}.$$

The projection of the segment $\overline{\mathsf{AB}}$ on the unit sphere has the following parametric equations:

$$\mathsf{Q}(t) = \frac{\mathsf{P}(t)}{||\mathsf{P}(t)||}.$$

At this point there is nothing to do but use the already considered parallel projection form 3D to 2D to obtain a 2D curve. The final result for the football is shown in figure 16.

One last practical tip: the Ti*k*Z code of a figure like figure 16 is very long and complex (about 250 rows!) and it is useful to export it from GeoGebra one piece at a time, and not all together, especially if you need to paint the different parts in different ways (in our figure only black sphere pentagons and white sphere hexagons). It will be simpler to correctly fill the various parts of the figure, or to check if everything works correctly.

Figure 16: The football obtained by the projection of the truncated icosahedron on the circumscribed sphere

A simple but interesting application of this technique is shown in figure 17 where we have projected on the circumscribed sphere the regular tetrahedron. This figure solves an interesting problem: is it possible to cut an apple into four equivalent parts in an uncommon way?

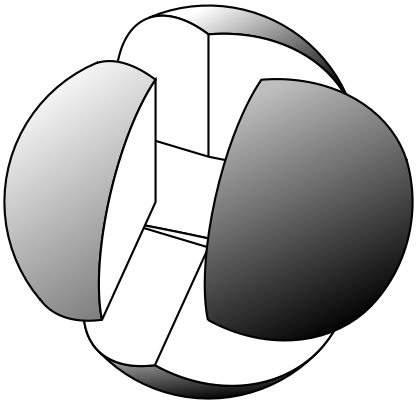


Figure 17: An apple cut in four parts in a non standard way

Before ending this "sport" section of our article we present a simple figure obtained from the truncated icosahedron: the molecule of the *Buckminsterfullerene*. In this case we have simply replaced the segments that make up the sides of the polyhedron by tubes and the vertices by shaded spheres. The following code for the tubes is taken from `https://tex.stackexchange.com`:

```
\newcommand{\Tube}[6][]%
{\colorlet{InColor}{#4}
 \colorlet{OutColor}{#5}
 \foreach \I in {1,...,#3}
 {\pgfmathsetlengthmacro{\h}{(\I-1)/#3*#2}
  \pgfmathsetlengthmacro{\r}{sqrt(pow(#2,2)
    -pow(\h,2))}
  \pgfmathsetmacro{\c}{(\I-0.5)/#3*100}
  \draw[InColor!\c!OutColor, line width=\r,#1]
     #6;
 }
}
```

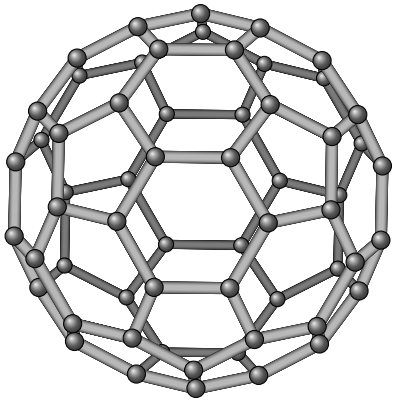


Figure 18: The molecule of Buckminsterfullerene

In a figure like this, in order to hide the invisible parts you need only to plot the rear parts first. As usual, you can easily locate them using the GeoGebra figure.

7 Conic and spherical sections

A very relevant problem for people interested in 3D graphics is the drawing of plots concerning conic sections. We only show some examples without extended details; the technique to be used is now familiar because, naturally, the involved curves are conics that GeoGebra can deal with using standard commands.

Figure 19 illustrates the two series of circular sections in an oblique cone; the sections parallel to the basis and the subcontrary sections, as considered by Apollonius. The complexity of this figure is due to mathematical calculations; you must find the correct angle for the plane that produces the subcontrary section, and the best way to do this is the original Apollonius description.

Figure 20 shows how to section a cone in order to obtain a hyperbola. The technique to obtain such a figure is simple in GeoGebra; after plotting the entire cone and the hyperbola on the cone you can hide, directly in GeoGebra, one of the two parts, leaving only the remaining one. After exporting the code you can shift, for instance, the right part using the following very standard code:

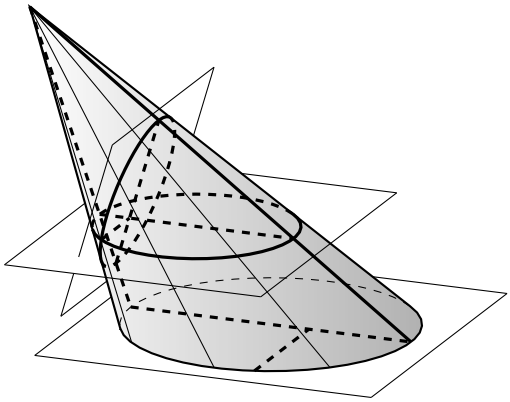


Figure 19: The two series of circular sections that can be obtained in an oblique cone

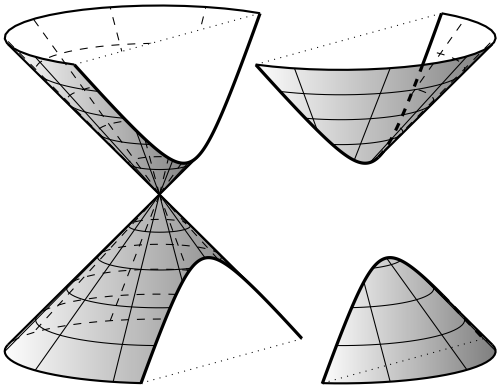


Figure 20: Section of a cone to obtain a hyperbola

```
\begin{scope}[xshift=2.4cm]
⟨code of the second part⟩
\end{scope}
```

The last figure of this section, figure 21, is somewhat more complicated, because GeoGebra can't handle directly (at least at the moment) the intersection between a cylinder and a sphere. Anyhow, the intersection curve is a Vivianis's window and the parametric equations can be found easily in all books of curves. The rest of the construction does not require special attention; there are only parts of a cylinder and of a sphere.

8 Some more advanced images

The technique based on exports from GeoGebra can also handle more complicated figures, but, naturally, a somewhat advanced knowledge of GeoGebra is required for this. In our opinion the effort is worth the candle because what you can obtain is very interesting. We give three images as examples.

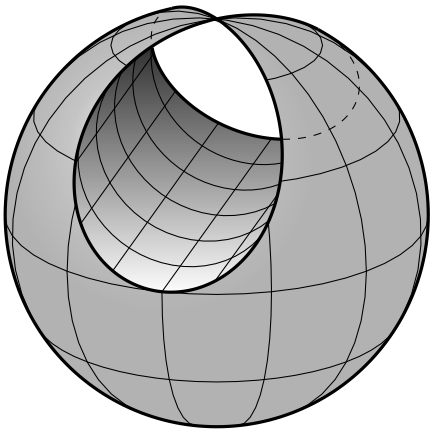


Figure 21: Intersection between a cylinder and a sphere

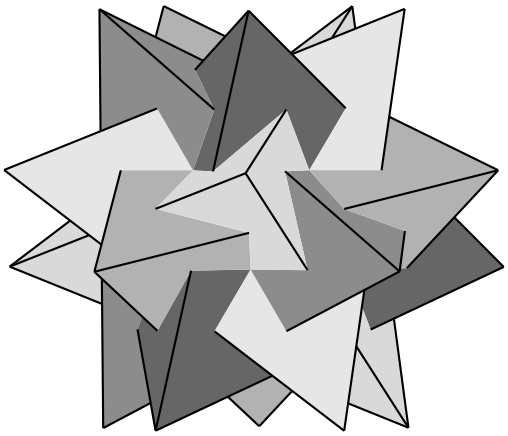


Figure 22: The compound of five tetrahedra

The first figure is the *compound of five tetrahedra*, which is one of the five regular polyhedral compounds. It can be constructed by arranging five tetrahedra inside a dodecahedron, having no common vertex. The correct construction of such a figure requires full attention, in particular to understand which are the actual sides and which instead are only fake sides, that must not be highlighted in the figure. Furthermore, in this case it is better to hide completely the non-visible part of the figure.

The second and third figures are reproductions of originals by Kepler, published in the 1596 in *Mysterium Cosmographicum*. They deal with a picture concerning the solar system as known in those times and consist of the five Platonic solids inscribed one into the other, while the inscribed/circumscribed spheres to each polyhedron contain the orbits of the six planets, earth included, with the sun at the center. As in the original by Kepler we propose both

the set of all the Platonic solids and a detail of the four interior spheres with the corresponding three polyhedra.

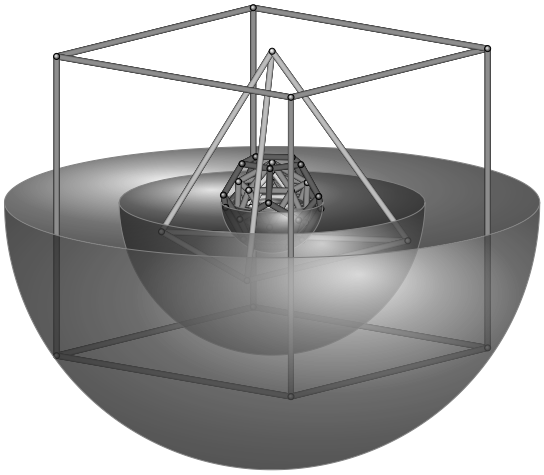


Figure 23: Reproduction of the solar system, as originally drawn by Kepler in 1596

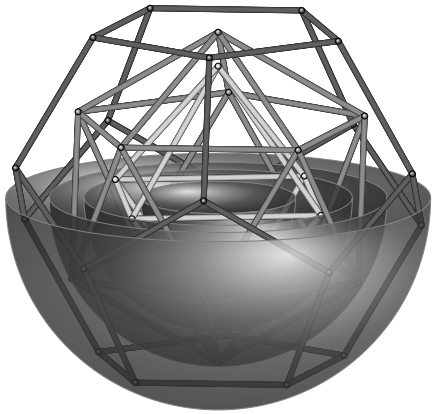


Figure 24: Reproduction of the solar system originally drawn by Kepler in 1596: detail of the central part

As with figure 18, one of the secrets for drawing correctly is to start from the back and to end by drawing the front parts of the figure.

9 Conclusion

We believe that the proposed technique can be advantageously used for the production of a large part of the geometric type figures required in a mathematical paper. This technique paired with pgf-plots or the corresponding packages for the PSTricks family allows the production of complex scientific books using LaTeX and without any external software.

As already mentioned, particularly when drawing complex figures, a somewhat deep knowledge of GeoGebra is required, but, in our experience, the learning curve of GeoGebra is much flatter than that of Ti*k*Z; the use of GeoGebra also offers the numerous advantages we have described in this article.

Naturally there is no rose without thorns and it is not possible to achieve the effects we have described without hard work and experimentation.

References

[1] Luciano Battaia. Grafica 3D con Geogebra e Ti*k*Z. *ArsTEXnica*, 22:50–63, 2016. guitex.org/home/images/ArsTeXnica/AT022/battaia.pdf.

[2] Claudio Beccari. The unknown picture environment. *ArsTEXnica*, 11:57–64, 2011. tug.org/TUGboat/tb33-1/tb103becc-picture.pdf.

[3] Agostino De Marco. Illustrazioni tridimensionali con Sketch/LaTeX/PSTricks/Ti*k*Z nella didattica della Dinamica del Volo. *ArsTEXnica*, 4:51–68, 2007. guitex.org/home/it/numero-4.

[4] Tomasz M. Trzeciak. texample.net/tikz/examples/map-projections/, 2008.

[5] Keith Wolcott. Three-dimensional graphics with PGF/Ti*k*Z. *TUGboat*, 33(1):102–113, 2012. tug.org/TUGboat/tb33-1/tb103wolcott.pdf.

◇ Luciano Battaia
Via Garibaldi 4
San Giorgio Richinvelda, PN 33095
Italy
luciano.battaia (at) unive dot it
http://www.batmath.it

ConTEXt nodes: commutative diagrams and related graphics

A. Braslau, I. Hamid, and H. Hagen

Abstract

The graphical representation of node-based textual diagrams is a very useful tool in the communication of ideas. These are composed of graphical objects or blocks of text or a combination of both, i.e. a decorated label or text block, each attached to some point (= the node). Additionally, such diagrams may display other such objects (such as a line segment, an arrow, or other curve) connecting node points. The set of nodes of a diagram will have some spatial relation between nodes. In this article we discuss a new MetaPost module for handling node-based graphics, as well as a derivative simplified ConTEXt module.

1 Introduction

The graphical representation of textual diagrams is a very useful tool in the communication of ideas. In category and topos theory, for example, many key concepts, formulas, and theorems are expressed by means of *commutative diagrams*; these involve objects and arrows between them. Certain concepts discovered by category theory, such as *natural transformations*, are becoming useful in areas outside of mathematics and natural science, e.g., in philosophy. To make category and topos methods usable by both specialists and non-specialists, commutative diagrams are an indispensable tool. (For many examples of formal and informal commutative diagrams, see [1].) The use of nodal diagrams is not limited to category theory: they may represent a flow diagram (of a process, for example), a chemical reaction sequence or pathways, phases and phase transitions, a hierarchical structure (of anything), a timeline or sequence of events or dependencies, a family tree, etc.

The basic units of a node-based diagram include *node objects*, each attached to some point (= the *node*) in some spatial relationship. Note that a set of objects might be associated with a single node. Given a node, it also stands in a spatial relation to some other node. The spatial relationship between the set of nodes of a diagram need not be in a regular network, although it often is. Note that the spatial relationship between nodes is graphical and may represent, e.g., a temporal or logical relationship, or a transformation of one object into another or into others (one interesting example might be that representing cell division or mitosis).

Given a spatial relation between any two nodes, a node-based diagram often includes some *path segment* or segments (such as arrows or other curves) between two given nodes that *relate(s)* them. Each path segment may be augmented by some textual or graphical label.

A simple example of a node diagram is shown in Figure 1.

Figure 1

More precisely, a *node* is a point of intersection or branching of paths, often a point on a regular lattice. (The nodes of the above diagram are the two endpoints of a straight line segment.) Sometimes, however, a node might be a single point as in an *identity map* of category theory, referring to itself:

Figure 2

> The standard arrowhead in MetaPost is a simple triangle, whose length and angle can be adjusted. Metafun provides further options, allowing this arrowhead to be barbed or dimpled. In the present article, we use the settings `ahlength:=10pt; ahangle:=30; ahvariant:=1; ahdimple:=4/5;`. The loop-back arrow paths used here deviate from a circular segment, becoming ellipsoidal, through the value `node_loopback_yscale:=.7;`. These are all set within a `\startMPinitializations ... \stopMPinitializations` pair.

In this article we discuss a new MetaPost module designed for handling node-based graphics as well as a derivative simple ConTEXt interface. To illustrate, the code producing A⟶B could be, in MetaPost and the ConTEXt interface respectively:

– MetaPost:

```
draw node(0,"A") ;
draw node(1,"B") ;
drawarrow fromto(0,1) ;
```

The MetaPost code shown here has been simplified, as will be seen further on.

– ConTEXt:

```
\startnodes [dx=1.5cm]
  \placenode [0,0] {A}
  \placenode [1,0] {B}
  \connectnodes [0,1] [alternative=arrow]
\stopnodes
```

The ConTEXt interface has a limited set of features, and will remain simple.

In each case, TEX is told to draw an arrow from A to B (i.e., from node 0 to node 1).

For beginners, casual users of ConTEXt, or any

others who might be intimidated by MetaPost syntax, the ability to construct simple diagrams by means of standard ConTEXt syntax is helpful. For those who have tried the ConTEXt interface and/or want to draw more advanced diagrams, the MetaPost module is much more powerful and flexible.

2 MetaPost

MetaPost is a vector-graphics language which calls upon TEX to typeset text (such as labels); in ConTEXt, furthermore, MetaPost is integrated natively through the library MPlib as well as the macro package Metafun. The tight integration of ConTEXt and MetaPost provides advantages over the use of other, external graphics engines. These advantages include ease of maintaining coherence of style, as well as extensive flexibility without bloat. MetaPost has further advantages over most other graphics engines, including a very high degree of precision as well as the possibility to solve certain types of algebraic equations. This last feature is rarely used but should not be overlooked.

It is quite natural in MetaPost to locate our node objects along a path or on differing paths. This is a much more powerful concept than merely locating a node at some pair of coordinates, e.g., on a square or a rectangular lattice, for example (as in a table). Furthermore, these paths may be in three dimensions (or more); of course the printed page will only involve some projection onto two dimensions. Nor are the nodes restricted to a location on the points defining a path: they may have, for an index, any *time* along a given path `p` ranging from the first defining point ($t = 0$) up to the last point of that path ($t \leq$ `length(p)`), the number of defining points of a path. (Note that the time of a cyclic path is taken modulo the length of the path, that is, t outside of the range [`0, length(p)`] will return the first or the last point of an open path, but will "wrap" for a closed path.)

Given a path `p`, nodes are defined (implicitly) as `picture` elements: `picture p.pic[];`. This is a pseudo-array where the square brackets indicate a set of numerical tokens, as in `p.pic[0]` or `p.pic[i]` (for `i=0`), but also `p.pic0`. This number need not be an integer, and `p.pic[.5]` or `p.pic.5` (not to be confused with `p.pic5`) are also valid. These picture elements are taken to be located relative to the path `p`, with the index `t` corresponding to a time along the path, as in

```
draw p.pic[t] shifted point t of p;
```

(although it is not necessary to draw them in this way). This convention allows the nodes to be oriented and offset with respect to the path in an arbitrary manner.

Note that a path can be defined, then nodes placed relative to this path. Or the path may be declared but remain undefined, to be determined only after the nodes are declared. Yet another possibility is that the path may be adjusted as needed, as a function of whatever nodes are to be occupied. This will be illustrated through examples further down.

3 Some simple examples

Let's begin by illustrating a typical commutative diagram from category theory. Although it may appear trivial, this example helps to introduce MetaPost syntax. At the same time, a large part of the idea behind this module is to facilitate use of this system without having to learn much MetaPost.

```
\startMPcode
  path p ; p := fullsquare scaled 3cm ;
  draw p ;
  for i=0 upto length p:
    draw point i of p
      withpen pencircle scaled 5pt ;
  endfor ;
\stopMPcode
```

A path is drawn as well as the points defining the path.

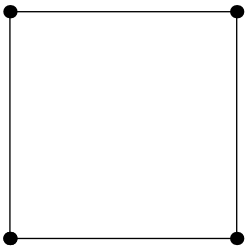


Figure 3

Given the named path `nodepath`, we can now define and draw nodes as well as connections between them (see Figure 4):

```
\startMPcode
  clearnodepath ; nodepath = p ;
  draw node(0,"\node{$G(X)$}") ;
  draw node(1,"\node{$G(Y)$}") ;
  draw node(2,"\node{$F(Y)$}") ;
  draw node(3,"\node{$F(X)$}") ;
  drawarrow fromto.bot(0,0,1,
         "\nodeSmall{$G(f)$}") ;
  drawarrow fromto.top(0,3,2,
         "\nodeSmall{$F(f)$}") ;
  drawarrow fromto.rt (0,2,1,
         "\nodeSmall{$\eta_Y$}") ;
  drawarrow fromto.lft(0,3,0,
         "\nodeSmall{$\eta_X$}") ;
\stopMPcode
```

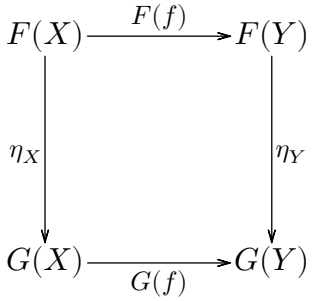


Figure 4 Drawn using the MetaPost interface.

In working with MetaPost, it is good practice to reset or clear a variable using the directive `save` for the *suffix* (or variable name) `nodepath`, as contained in the directive `clearnodepath` (defined as "`save nodepath; path nodepath`"). The macros used here rely on the creation of certain internal variables and may not function correctly if the variable structure is not cleared. Indeed, any node may contain a combination of picture elements, added successively, so it is crucial to `save` the variable, making its use local rather than global. This point is particularly true with ConTEXt, where a single MPlib instance is used and maintained over multiple runs.

> The ConTEXt directives `\startMPcode...\stopMPcode` include grouping (MetaPost `begingroup;...endgroup;`) and the use of `save` (in `clearnodepath`) will make the suffix `nodepath` local to this code block. In the code for Figures 3 and 4, the path `p` itself is not declared local (through the use of a `save`); it therefore remains available for other MetaPost code blocks. We cannot do this with the default suffix name `nodepath` without undesirable consequences.

Note that one should not confuse the above MetaPost function `node()` with the ConTEXt command `\node{}`, defined as follows:

```
\defineframed
  [node]
  [frame=off,
   offset=1pt]

\defineframed
  [nodeSmall]
  [node]
  [foregroundstyle=small]
```

`\node{}` places the text within a ConTEXt frame (with the frame border turned off), whereas the MetaPost function `node(i,"...")` sets and returns a picture element associated with a point on path `nodepath` indexed by its first argument. The second argument here is a string that gets typeset by TEX. (The use of `\node{}` adds an `offset`.)

By default, the MetaPost function `fromto()` returns a path segment going between two points of the path `nodepath`. The first argument (`0` in the example above) can be used as a displacement to skew the path away from a straight line (by an amount in units of the straight path length). The last argument is a string to be typeset and placed at the midpoint of the segment. The suffix appended to the function name gives an offset around this halfway point. This follows standard MetaPost conventions.

It is important to draw or declare the nodes *before* drawing the connections, using `fromto()`, in order to avoid overlapping symbols, as one notices that the arrows drawn in the example above begin and end on the border of the frame (or bounding box) surrounding the node text. This would not be possible if the arrow were to be drawn before this text was known.

As will be seen further on, one can specify the use of any defined path, without restriction to the built-in name `nodepath` that is used by default. Furthermore, a function `fromtopaths()` can be used to draw segments connecting any two paths which may be distinct. This too will be illustrated further on.

The ConTEXt syntax for the current example looks like this:

```
\startnodes [dx=3cm,dy=3cm]
  \placenode [0,0] {\node{$G(X)$}}
  \placenode [1,0] {\node{$G(Y)$}}
  \placenode [1,1] {\node{$F(Y)$}}
  \placenode [0,1] {\node{$F(X)$}}
  \connectnodes [0,1] [alternative=arrow,
    label={\nodeSmall{$G(f)$}},position=bottom]
  \connectnodes [3,2] [alternative=arrow,
    label={\nodeSmall{$F(f)$}},position=top]
  \connectnodes [2,1] [alternative=arrow,
    label={\nodeSmall{$\eta_Y$}},position=right]
  \connectnodes [3,0] [alternative=arrow,
    label={\nodeSmall{$\eta_X$}},position=left]
\stopnodes
```

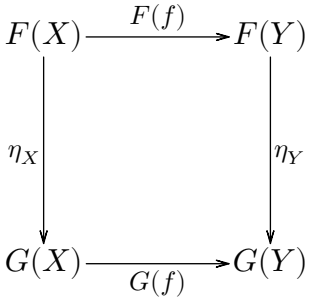


Figure 5 Drawn using the ConTEXt interface.

This follows the more classic (and limited) approach of placing nodes on the coordinates of a regular lattice, here defined as a 3 cm square network. [The lattice can be square (`dx` = `dy`), rectangular (`dx` ≠ `dy`), or oblique (through `rotation` ≠ 90).] The arguments are then (x, y) coordinates of this lattice and the nodes are indexed 0, 1, 2, ... in the order in which they are drawn. These are used as reference indices in the commands `\connectnodes` (rather than requiring two *pairs* of coordinates); see Figure 6.

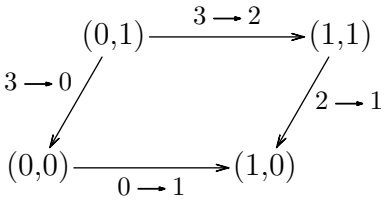


Figure 6 Coordinates and indices. (For variety, a rectangular oblique lattice is drawn.)

Connecting numbered nodes (in the order in which they were declared) might seem a bit confusing at first view, but it simplifies things in the end, really!

An identity map, as shown in Figure 2, earlier, and, below, in Figure 7 is achieved by connecting a node to itself.

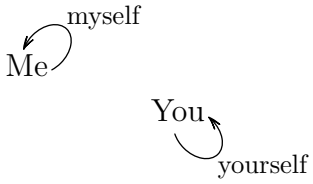


Figure 7 Identity maps

```
\startnodes [dx=2cm,dy=.6cm]
  \placenode [0,0]  {\node{Me}}
  \placenode [1,-1] {\node{You}}
  \connectnodes [0,0] [alternative=arrow,
    offset=.75cm,position=topright,
    label=myself]
  \connectnodes [1,1] [alternative=arrow,
    offset=.75cm,position=bottomright,
    label=yourself]
\stopnodes
```

The scale (diameter) of the circular loop-back is set by the keyword `offset=` (normally used to curve or bow-away a path connecting nodes from the straight-line segment between them), and the `position=` keyword sets its orientation.

Let us now consider the following code which illustrates the Metafun operator `crossingunder` (see Figure 8). The `nodepath` indices are put into variables `A`, `B`, `C`, and `D`, thus simplifying the code.

```
\startMPcode
  clearnodepath ;
  nodepath := fullsquare scaled 2cm ;
  save A,B,C,D ;
  A = 3 ; draw node(A,"\node{A}") ;
  B = 2 ; draw node(B,"\node{B}") ;
  C = 0 ; draw node(C,"\node{C}") ;
  D = 1 ; draw node(D,"\node{D}") ;
  drawarrow fromto(0,B,C) ;
  drawarrow fromto(0,A,D)
    crossingunder fromto(0,B,C) ;
\stopMPcode
```

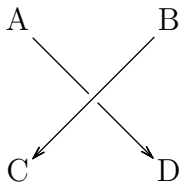


Figure 8 A→D under B→C.

Given a path segment to be crossed, `crossingunder` draws a path with a segment surrounding the intersection with that path cut-out, resulting in two (sub)path segments. This operator is of such general use that it has been added to the Metafun base.

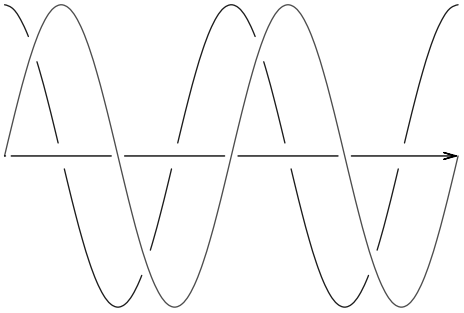


Figure 9 `crossingunder`

Another illustration of the `crossingunder` operator in use is shown in figure 9. Because the diagrams are all defined and drawn in MetaPost, one can easily use the power of MetaPost to extend a simple node drawing with any kind of graphical decoration.

This brings up an important point that has limited the development of a full-featured ConTEXt node module up to now. A pure MetaPost interface affords much more flexibility than can be conveniently reduced to a set of TEX macros; the ConTEXt interface has been written to provide only basic functionality. (One can use `\nodeMPcode{}` to inject arbitrary MetaPost code within a `\startnode...\stopnode` pair, although in this example one is probably better off using the straight MetaPost interface.)

4 Cyclic diagrams

For a somewhat more complicated example, let us consider the representation of a catalytic process such as that given by Krebs [2]. The input is shown coming into the cycle from the center of a circle; the products of the cycle are spun off from the outside of the circle. We start by defining a circular path where each point corresponds to a step in the cyclic process. Our example will use six steps.

We also want to define a second circular path with the same number of points at the interior of this first circle for the input, and a third circular path at the exterior for the output (see Figure 10).

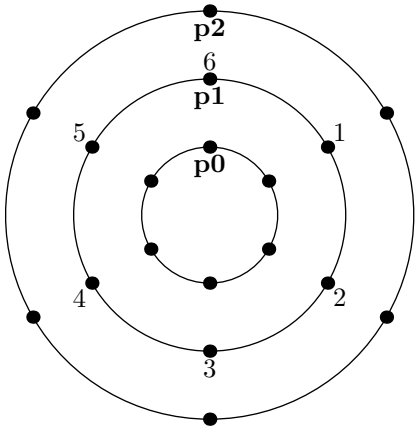


Figure 10 The paths that we will use for the anchoring of nodes.

The code is as follows:

```
\startMPcode
  save p ; path p[] ;
  % define a fullcircle path
  % with nodes at 60° (rather than 45°)
  p1 := (for i=0 step 60 until 300:
         dir(90-i) .. endfor cycle)
        scaled 1.8cm ;
  p0 := p1 scaled  .5 ;
  p2 := p1 scaled 1.5 ;

  for i=0 upto 2:
    draw p[i] ;
    for j=1 upto length p[i]:
      draw point j of p[i]
        withpen currentpen scaled 10 ;
      if i=1:
        label.autoalign(angle
                        point j of p[i])
          (decimal j, point j of p[i]) ;
      fi
    endfor
    label.bot("\bf p" & decimal i,
              point 0 of p[i]) ;
  endfor
\stopMPcode
```

(`autoalign()` is a feature defined within Metafun.)

Nodes will then be drawn on each of these three circles and arrows will be used to connect the various nodes, either on the same path or else between paths.

The MetaPost function `fromto()` is used to give a path segment that points from one node to another. It *assumes* the path named `nodepath`, and in fact calls the function `fromtopaths` that explicitly takes path names as arguments. That is, `fromto (d, i, j, ...)` is equivalent to `fromtopaths (d, nodepath, i, nodepath, j, ...)`.

As stated above, this segment can be a straight line, or a path can be bowed away from this straight line by a transverse displacement given by the function's first argument (given in units of the straight segment length). When both nodes are located on a single, defined path, this segment can be made to lie on or follow this path, such as one of the circular paths defined above. This behavior is obtained by using any non-numeric value (such as `true`) in place of the first argument. Of course, this cannot work if the two nodes are not located on the same path.

In figure 11, the circular arc segments labeled *a*–*f* are drawn using the following:

```
drawarrow fromtopaths.urt
          (true,p1,0,p1,1,"\nodeGray{a}") ;
```

Here, `\nodeGray` is a frame that inherits from `\node`, changing style and color:

```
\defineframed
  [nodeGray]
  [node]
  [foregroundcolor=darkgray,
   foregroundstyle=italic]
```

The bowed arrows feeding into the cyclic process and leading out to the products — between different paths, from the path `p0` to the path `p1` and from the path `p1` to the path `p2`, respectively — are drawn using the deviations `+3/10` and `-1/10` (to and from half-integer indices, thus mid-step, on path `p1`):

```
drawarrow fromtopaths( 3/10,p0,0,p1,0.5)
          withcolor .6white ;
drawarrow fromtopaths(-1/10,p1,0.5,p2,1)
          withcolor .6white ;
```

4.1 A lesson in MetaPost

An 'array' of paths is declared through `path p[];` it is not a formal array, but rather a syntactic definition of a collection of path variables `p0`, `p1`, ..., each of whose names is prefixed with the tag "p" followed by any number, not necessarily an integer (e.g., `p3.14` is a valid path name). The syntax allows enclosing this "index" within square brackets, as in `p[0]` or, more typically, `p[i]`, where `i` would be a numeric

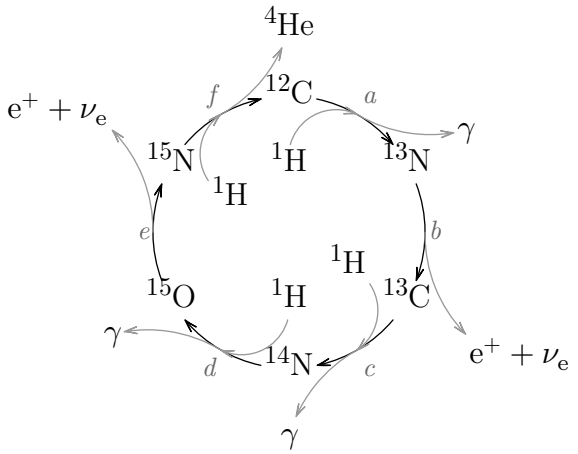


Figure 11 The Bethe cycle for the energy production in stars [3] in a Krebs representation of a catalytic process [2].

variable or the index of a loop. Note that the use of brackets is required when using a negative index, as in `p[-1]` (since `p-1` is interpreted as three tokens, representing a subtraction). Furthermore, the variable `p` itself would here be a numeric (by default), so `p[p]` would be a valid syntactic construction! One could, additionally, declare a set of variables `path p[][];` and so forth, defining also `p[0][0]` (equivalently, `p0 0`) for example as a valid path, coexisting with yet different from the path `p0`.

MetaPost also admits variable names reminiscent of structured types in programming; for example, the declaration `picture p.pic[];` is used internally in the `node` macros, but this becomes `picture p[]pic[];` when using a path 'array' syntax. These variable names are associated with the suffix `p` and all become undefined by `save p;`.

5 Putting it together

What follows is an example of a natural transformation, discovered and articulated in the course of a philosophical research project (by Idris Samawi Hamid). Figure 12 represents what is called the Croce Topos, named after the Italian philosopher Benedetto Croce (1866–1952). We define it using the ConTEXt interface to the node package:

```
\startnodes [dx=3cm,dy=3cm,alternative=arrow]
  \placenode [0,  0] {\node{Practical}}
  \placenode [1,  0] {\node{Economic}}
  \placenode [3.5,0] {\node{Moral}}
  \placenode [3.5,1] {\node{Conceptual}}
  \placenode [1,  1] {\node{Aesthetic}}
  \placenode [0,  1] {\node{Theoretical}}
  \connectnodes [5,0] [offset=.1,
    position=right,label={\node{$\gamma$}}]
  \connectnodes [0,5] [offset=.1,
    position=left, label={\node{$\gamma'$}}]
  \connectnodes [4,1] [offset=.1,
    position=right,label={\node{$F\gamma$}}]
  \connectnodes [1,4] [offset=.1,
    position=left, label={\node{$F\gamma'$}}]
  \connectnodes [3,2] [offset=.1,
    position=right,label={\node{$G\gamma$}}]
  \connectnodes [2,3] [offset=.1,
    position=left, label={\node{$G\gamma'$}}]
  \connectnodes [4,3] [position=top,
    label={\node{\it concretization$_1$}}]
  \connectnodes [3,4] [position=bottom,
    offset=.1,option=dashed,
    label={\node{\it abstraction$_1$}}]
  \connectnodes [1,2] [position=top,
    label={\node{\it concretization$_2$}}]
  \connectnodes [2,1] [position=bottom,
    offset=.1,option=dashed,
    label={\node{\it abstraction$_2$}}]
\stopnodes
```

6 Tree diagrams

The tree diagram shown in Figure 13 is drawn using four paths, each one defining a row or generation in the branching. The definition of the spacing of nodes was crafted by hand and is somewhat arbitrary: 3.8, 1.7, and 1 for the first, second and third generations. This might not be the best approach, but it is how I (Alan) was thinking when I first created this figure.

Ultimately, one can do better by allowing MetaPost to solve the relevant equations and determine this spacing automatically. Because this is a somewhat advanced procedure, this approach will be first illustrated through a simple example of a diagram where the nodes will be placed on a declared but undefined path:

```
save p ; % path p ;
```

The `save p;` assures that the path is undefined. This path will later be defined based on the contents of the nodes and a desired relative placement.

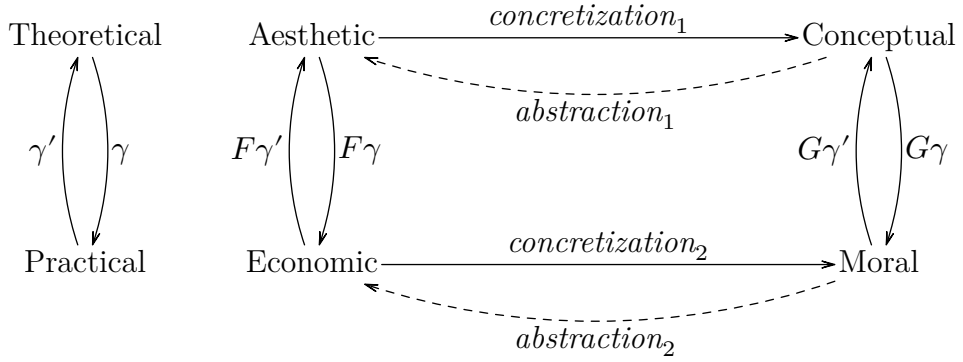


Figure 12 A representation of the Croce Topos.

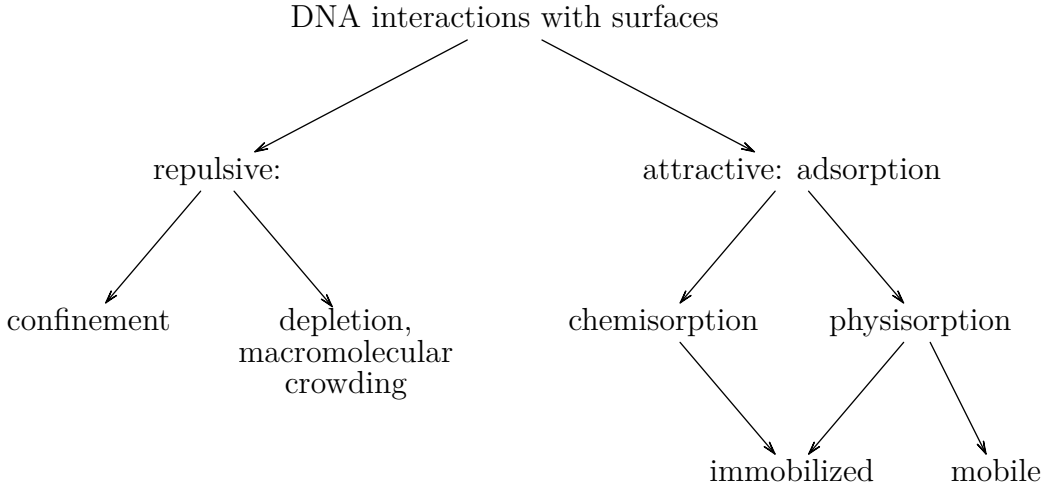


Figure 13 An example tree diagram.

In fact, it is not even necessary to declare that the suffix will be a path, as the path will be declared and automatically built once the positions of all the nodes are determined. To emphasize this point, the `path` declaration above is commented out.

Warning: Solving equations in MetaPost can be non-trivial for those who are less mathematically inclined. One needs to establish a coupled set of equations that is solvable: that is, fully but not over-determined.

A few helper functions have been defined: `makenode()` returns a suffix (variable name) corresponding to the node's position. The first such node can be placed at any finite point, for example the drawing's origin. The following nodes can be placed in relation to this first node:

```
save nodepath ;
save first, second, third, fourth ;
pair first, second, third, fourth ;
first.i = 0 ; second.i = 1 ;
third.i = 2 ; fourth.i = 3 ;
first  = makenode(first.i, "\node{first}");
second = makenode(second.i,"\node{second}");
third  = makenode(third.i, "\node{third}");
fourth = makenode(fourth.i,"\node{fourth}");

first  = origin ;
second = first
       + betweennodes.urt(nodepath,first.i,
                          nodepath,second.i,
                          whatever) ;
third  = second
       + betweennodes.lft(nodepath,second.i,
                          nodepath,third.i,
                          whatever) ;
fourth = third
       + betweennodes.bot(nodepath,fourth.i,
                          nodepath,first.i,
                          3ahlength) ;
```

The helper function `betweennodes()` returns a vector pointing in a certain direction (here following the standard MetaPost suffixes `urt`, `lft`, and `bot`), that takes into account the bounding boxes of the contents of each node, plus an (optional) additional distance (here given in units of the arrow-head length, `ahlength`). Using the keyword `whatever`

tells MetaPost to adjust this distance as necessary. The above set of equations is incomplete as written, so a fifth and final relation needs to be added; the fourth node is also to be located directly to the left of the very first node:

```
fourth = first
       + betweennodes.lft(nodepath,fourth.i,
                          nodepath,first.i,
                          3ahlength) ;
```

(Equivalently, we could declare that the first node located to the right of the fourth node: `first = fourth + betweennodes.rt (nodepath, first.i, nodepath, fourth.i, 3ahlength);`.)

Note that the helper function `makenode()` can be used as many times as needed; if given no content, it merely returns the node's position. Additional nodes can be added to this diagram along with appropriate relational equations, keeping in mind that the equations must, of course, be solvable. This last issue is the one challenge that most users might face.

The function `node()`, used previously and returning a picture element to be drawn, itself calls the function `makenode()`, used here. The nodes have not yet been drawn:

```
for i = first.i,second.i,third.i,fourth.i :
  draw node(i) ;
  drawarrow fromto(0,i,i+1) ;
endfor
```

This results in Figure 14. The path is now defined as one running through the position of all of the defined nodes, and is cyclic.

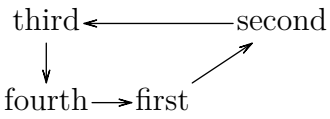


Figure 14

Using this approach, that of defining but not drawing the nodes until a complete set of equations defining their relative positions has been constructed, imposes several limitations. First, the nodes are expected to be numbered from 0 to n, continuously and without any gaps for each defined path. This is just an implicit, heuristic convention of the path construction. Second, when ultimately defining all the nodes and their positions, the path needs to be constructed. A function, `makenodepath(p)` accomplishes this; it gets implicitly called (once) upon the drawing of any `node()` or connecting `fromto`. Of course, `makenodepath()` can always be called explicitly once the set of equations determining the node positions is completely defined.

We once again stress that the writing of a solvable, yet not over-determined, set of equations can be a common source of error for many MetaPost users.

Another example is the construction of a simple tree of descendance, a.k.a. a family tree. There are many ways to draw such a tree; in Figure 15, we show only three generations. We leave it as an exercise to the reader to come up with the equations used to determine this tree (or one can look at the source of this document).

The requisite set of equations could be hidden from users wishing to construct simple, pre-defined types of diagrams. However, such cases would involve a loss of generality and flexibility. Nevertheless, the ConTEXt-Nodes module *could* be extended in the future to provide a few simple models. One might be a branching tree structure, although even the above example (as drawn) does not easily fit into a simple, general model.

A user on the ConTEXt mailing list asked if it is possible to make structure trees for English sentences with categorical grammar, an example of which is shown in Figure 16.

Here, I chose to define a series of parallel paths, one per word, with one path terminating whenever it joins another path (or paths) at a common parent. Naturally, labeling each branch of the tree structure requires a knowledge of the tree structure. The code is not short, but hopefully it is mostly clear. Note that diagrams such as those constructed here will each be significantly different, making the writing of a general mechanism rather complex. For example, one might need to construct a tree branching up rather than down, or to the right (or left), or even following an arbitrary path, such as a random walk. These can all be achieved individually in MetaPost without too much difficulty.

```
\startMPcode
  save p ; path p[] ;
  save n ; n = 0 ;
  % rather than parsing a string,
  % we can use "suffixes":
  forsuffixes $=People,from,the,country,
                can,become,quite,lonely :
    p[n] = makenode(p[n],0,
           "\node{\it" & (str $) & "}")
         = (n,0) ;
         % we work first with unit paths.
    n := n + 1 ;
  endfor
  save u ; u := MakeupWidth/n ;

  % build upward tree

  vardef makeparentnode(text t) =
```

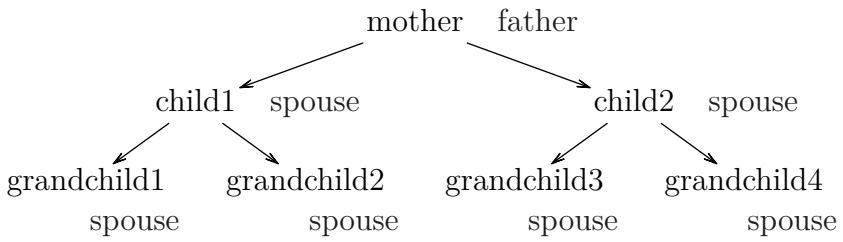


Figure 15 A tree of descendance.

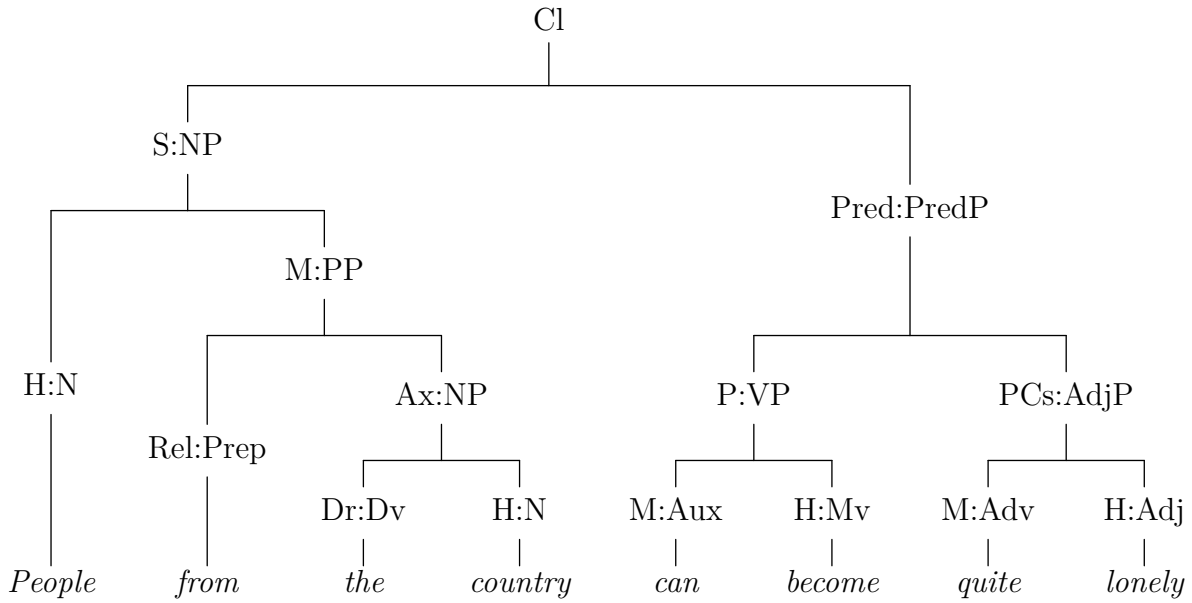


Figure 16 A categorical grammar structure tree.

```
  save i, xsum, xaverage, ymax ;
  i = xsum = 0 ;
  forsuffixes $ = t :
    clearxy ; z = point infinity of $ ;
    xsum := xsum + x ;
    if unknown ymax : ymax  = y ;
    elseif y > ymax : ymax := y ; fi
    i := i + 1 ;
  endfor
  xaverage = xsum / i ;
  ymax := ymax + 1 ;
  forsuffixes $ = t :
    clearxy ;
    z = point infinity of $ ;
    $ := $ & z -- (x,ymax)
      if i>1 : -- (xaverage,ymax) fi ;
  endfor
enddef ;

makeparentnode(p2,p3) ;
makeparentnode(p4,p5) ;
makeparentnode(p6,p7) ;
makeparentnode(p1,p2) ;
makeparentnode(p0,p1) ;
makeparentnode(p4,p6) ;
makeparentnode(p0,p4) ;
makeparentnode(p0) ;

% the paths are all defined
% but need to be scaled.

for i=0 upto n-1 :
  p[i] := p[i] xyscaled (u,.8u) ;
  draw node(p[i],0) ;
endfor

save    followpath ;
boolean followpath ; followpath = true ;

draw fromtopaths(followpath,p0,0,p0,1,
                          "\node{H:N}") ;
draw fromtopaths(followpath,p1,0,p1,1,
                       "\node{Rel:Prep}") ;
draw fromtopaths(followpath,p2,0,p2,1,
                         "\node{Dr:Dv}") ;
draw fromtopaths(followpath,p3,0,p3,1,
                          "\node{H:N}") ;
draw fromtopaths(followpath,p4,0,p4,1,
                         "\node{M:Aux}") ;
draw fromtopaths(followpath,p5,0,p5,1,
                          "\node{H:Mv}") ;
```

```
  draw fromtopaths(followpath,p6,0,p6,1,
                          "\node{M:Adv}") ;
  draw fromtopaths(followpath,p7,0,p7,1,
                          "\node{H:Adj}") ;

  draw fromtopaths(followpath,p1,1,p1,2) ;
  draw fromtopaths(followpath,p2,3,p2,4) ;
  draw fromtopaths(followpath,p1,2,p1,3,
                           "\node{M:PP}") ;
  draw fromtopaths(followpath,p2,1,p2,2) ;
  draw fromtopaths(followpath,p3,1,p3,2) ;
  draw fromtopaths(followpath,p2,2,p2,3,
                          "\node{Ax:NP}") ;
  draw fromtopaths(followpath,p4,1,p4,2) ;
  draw fromtopaths(followpath,p5,1,p5,2) ;
  draw fromtopaths(followpath,p4,2,p4,3,
                           "\node{P:VP}") ;
  draw fromtopaths(followpath,p6,1,p6,2) ;
  draw fromtopaths(followpath,p7,1,p7,2) ;
  draw fromtopaths(followpath,p6,2,p6,3,
                       "\node{PCs:AdjP}") ;
  draw fromtopaths(followpath,p0,1,p0,2) ;
  draw fromtopaths(followpath,p1,3,p1,4) ;
  draw fromtopaths(followpath,p0,2,p0,3,
                           "\node{S:NP}") ;
  draw fromtopaths(followpath,p4,3,p4,4) ;
  draw fromtopaths(followpath,p6,3,p6,4) ;
  draw fromtopaths(followpath,p4,4,p4,5,
                     "\node{Pred:PredP}") ;
  draw node(p0,4.5,"\node{Cl}") ;
  draw fromtopaths(followpath,p0,3,p0,4.5);
  draw fromtopaths(followpath,p4,5,p4,6) ;
\stopMPcode
```

7 A 3D projection

Although MetaPost is a 2D drawing language, it can be easily extended to work in 3D. Several attempts have been made in the past ranging from simple to complicated. Here, we will take a simple approach.

The MetaPost language includes a triplet variable type, used to handle `rgb` colors (it also has a quadruplet type used for `cmyk` colors). We will use this `triplet` type to hold 3D coordinates. There is a separate ConTEXt module, entitled `three`, which creates a new MetaPost instance (also named `three`), which loads a set of macros that can be used to manipulate these triplet coordinates.

```
\usemodule [three]

\startMPcode{three}
...
\stopMPcode
```

For our purposes here, only one function is really necessary: `projection()`, which maps a 3D coordinate to a 2D projection on the page. This will not be a perspective projection having a viewpoint and a focus point, but rather a very simple oblique projection, useful for, e.g., pseudo-3D schematic drawings. The `Z` coordinate is taken to be `up` and the `Y` coordinate taken to be `right`, both in the plane of the paper. The third coordinate `X` is an oblique projection in a right-hand coordinate system.

Intended for schematic drawings, there is no automatic hidden-line removal nor effects like shading, and line crossings need to be handled manually (using `crossingunder` introduced previously). In Figure 17 we draw a simple cubical commutative diagram, with a node at each corner.

```
\startMPcode{three}
  save nodepath ;
  path nodepath ;
  nodepath = (projection Origin --
              projection (1,0,0) --
              projection (1,1,0) --
              projection (0,1,0) --
              projection (0,1,1) --
              projection (1,1,1) --
              projection (1,0,1) --
              projection (0,0,1) --
              cycle) scaled 5cm ;

  draw node(0, "\node
    {${\cal C}_{i\cal P}^{\mathrm{nt}}$}");
  draw node(1, "\node
    {${\cal C}_{i\cal G}^{\mathrm{nt}}$}");
  draw node(2, "\node
    {${\cal C}_{j\cal P}^{\mathrm{nt}}$}");
  draw node(3, "\node
    {${\cal C}_{j\cal G}^{\mathrm{nt}}$}");
  draw node(4,
    "\node{${\cal C}_{j\cal G}$}") ;
  draw node(5,
    "\node{${\cal C}_{j\cal P}$}") ;
  draw node(6,
    "\node{${\cal C}_{i\cal G}$}") ;
  draw node(7,
    "\node{${\cal C}_{i\cal P}$}") ;

  interim crossingscale := 30 ;
  drawdoublearrows fromto(0,0,1) ;
  drawdoublearrows fromto(0,1,2) ;
  drawdoublearrows fromto(0,2,3) ;
  drawdoublearrows fromto(0,3,0)
     crossingunder fromto(0,2,5) ;
  drawdoublearrows fromto(0,7,6) ;
  drawdoublearrows fromto(0,6,5) ;
  drawdoublearrows fromto.ulft(0,5,4,
                      "\node{$\tau_j$~}") ;
  drawdoublearrows fromto.top (0,7,4,
                         "\node{$\sigma$}") ;
  drawdoublearrows fromto.lrt(0,0,7,
         "\node{$Ψ^{\mathrm{nt}}$}")
     crossingunder fromto(0,6,5) ;
```

```
  drawdoublearrows fromto(0,1,6) ;
  drawdoublearrows fromto(0,2,5) ;
  drawdoublearrows fromto(0,3,4) ;
\stopMPcode
```

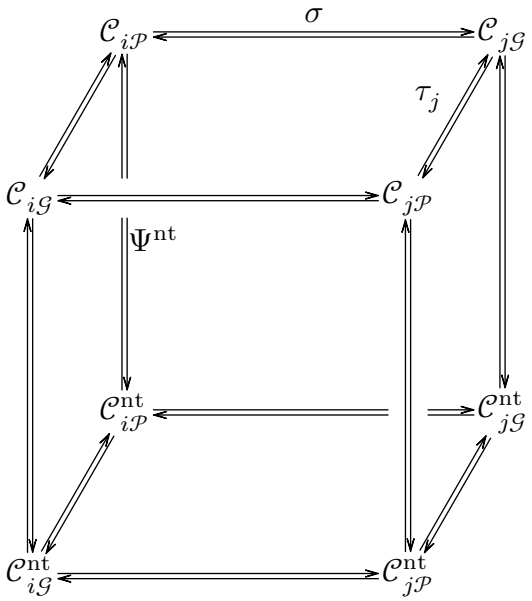


Figure 17

Note the use of `drawdoublearrows`, a new Metafun command that is introduced here.

8 Two final examples

We end this paper with two examples of more advanced commutative diagrams. The following example, shown in Figure 18, illustrates what in category theory is called a *pullback*. It is inspired from an example given in the Ti*k*Z CD (commutative diagrams) package.

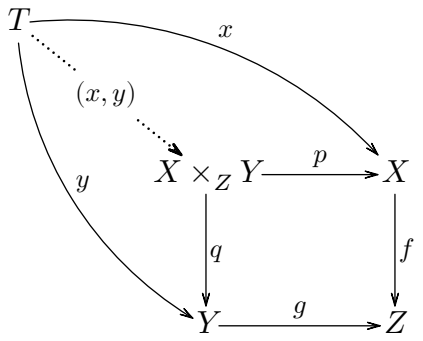


Figure 18

The arrow labeled "(x, y)" is drawn `dashed withdots` and illustrates how the line gets broken, implicitly `crossingunder` its centered label.

```
\startnodes [dx=2.5cm,dy=2cm,
             alternative=arrow]
  \placenode [0, 0] {\node{$X\times_Z Y$}}
  \placenode [1, 0] {\node{$X$}}
  \placenode [1,-1] {\node{$Z$}}
  \placenode [0,-1] {\node{$Y$}}
  \placenode [-1,1] {\node{$T$}}

  \connectnodes [0,1] [position=top,
    label={\nodeSmall{$p$}}]
  \connectnodes [1,2] [position=right,
    label={\nodeSmall{$f$}}]
  \connectnodes [0,3] [position=right,
    label={\nodeSmall{$q$}}]
  \connectnodes [3,2] [position=top,
    label={\nodeSmall{$g$}}]
  \connectnodes [4,0] [option=dotted,
    rulethickness=1pt,
    label={\nodeSmall{$(x,y)$}}]
  \connectnodes [4,1] [offset=+.13,
    position=top,
    label={\nodeSmall{$x$}}]
  \connectnodes [4,3] [offset=-.13,
    position=topright,
    label={\nodeSmall{$y$}}]
\stopnodes
```

The previous diagram was drawn using the ConTEXt interface. Our final example, shown in Figure 19, gives another "real-life" example of a categorical pullback, also inspired by Ti*k*Z-CD, but this time drawn through the MetaPost interface and solving for positions.

```
\startMPcode
  clearnodepath;
  save l ; l = 5ahlength ;
  save A, B, C, D, E ;
  pair A, B, C, D, E ;
  A.i = 0 ; B.i = 1 ; C.i = 2 ;
  D.i = 3 ; E.i = 4 ;
  A = makenode(A.i,"\node
      {$\pi_1(U_1\cap U_2)$}") ;
  B = makenode(B.i,"\node
      {$\pi_1(U_1)\ast_{\pi_1(U_1\cap U_2)}
                       \pi_1(U_2)$}") ;
  C = makenode(C.i,
      "\node{$\pi_1(X)$}") ;
  D = makenode(D.i,
      "\node{$\pi_1(U_2)$}") ;
  E = makenode(E.i,
      "\node{$\pi_1(U_1)$}") ;
  A = origin ;
  B = A + betweennodes.rt(nodepath,A.i,
                          nodepath,B.i)
        + (  1,0) ;
  C = B + betweennodes.rt(nodepath,B.i,
                          nodepath,C.i)
        + (.71,0) ;
  D = .5[A,B] + (0,-.91) ;
  E = .5[A,B] + (0, .91) ;
```

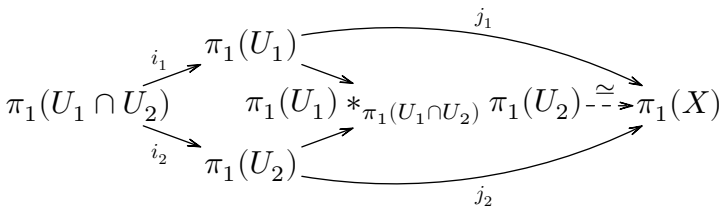


Figure 19 A categorical pullback diagram, with MetaPost finding the positions.

```
  for i = A.i, B.i, C.i, D.i, E.i :
    draw node(i) ;
  endfor
  drawarrow fromto.llft( 0,A.i,D.i,
              "\smallnode{$i_2$}") ;
  drawarrow fromto.ulft( 0,A.i,E.i,
              "\smallnode{$i_1$}") ;
  drawarrow fromto     ( 0,D.i,B.i) ;
  drawarrow fromto     ( 0,E.i,B.i) ;
  drawarrow fromto.urt( .1,E.i,C.i,
              "\smallnode{$j_1$}") ;
  drawarrow fromto.lrt(-.1,D.i,C.i,
              "\smallnode{$j_2$}") ;
  drawarrow fromto.top(  0,B.i,C.i)
    dashed evenly ;
  draw textext.top("{\strut$\simeq$}")
    shifted point .4 of fromto(0,B.i,C.i) ;
\stopMPcode
```

9 Conclusions

There was initial consensus at the 2017 ConTEXt Meeting in Maibach, Germany, where a version of this package was presented, that there was little use of developing a purely ConTEXt interface. Rather, the MetaPost package should be sufficiently accessible. Since then, however, we decided that the development of a derivative ConTEXt interface implementing some basic functionality could indeed be useful for many users, although it will necessarily remain somewhat limited. Users are recommended to turn to the pure MetaPost interface when more sophisticated functionality is needed.

10 Acknowledgements

This module was inspired by a request made by Idris Samawi Hamid to draw a natural transformation diagram in MetaPost (see Figure 4). The MetaPost macros that were then developed have benefited from improvements suggested by Hans Hagen as well as inspiration provided by Taco Hoekwater.

References

[1] F.W. Lawvere and S.H. Schanuel, *Conceptual Mathematics: A first introduction to categories* (2nd ed.), Cambridge University Press, Cambridge, UK, 2009.

[2] H.A. Krebs, "Cyclic processes in living matter", *Enzymologia* 12, 88–100, 1946.

[3] H.A. Bethe, "Energy Production in Stars", *Physical Review* 55, 103–103, 1939; H.A. Bethe, "Energy Production in Stars", *Physical Review* 55, 434–456, 1939.

◇ A. Braslau, I. Hamid, and H. Hagen
The ConTEXt development team
braslau.list (at) comcast.net,
ishamid (at) colostate.edu,
pragma (at) wxs.nl

TEX's "additional demerits" parameters

Udo Wermuth

Abstract

TEX has three integer parameters, `\adjdemerits`, `\finalhyphendemerits`, `\doublehyphendemerits`, which are added to the line demerits by the line-breaking algorithm if certain conditions are met. This article lists which lines of a paragraph can be affected by these parameters and presents some simple techniques to avoid their extensive application in a paragraph.

1 Introduction

The line-breaking procedure of TEX assigns to a set of line breaks for a paragraph a numerical value called its *demerits*; the higher this value the less desirable it is for TEX to use this set of line breaks to typeset the paragraph. The demerits of a paragraph are the sum of the demerits of each line. The calculation of these *line demerits* applies several concepts: The non-negative *badness* of the line, which is based on the width of the white spaces in the line, a penalty value that is added to the badness values, the *penalty* associated with the place at which the line is broken, and visual characteristics involving also the previous line. This last concept is realized by adding the integer values of three parameters, called the *additional demerits*.

The following formula calculates the line demerits, Λ_ι, for line number ι:

$$\Lambda_\iota = (\lambda + \beta_\iota)^2 + \mathrm{sgn}(\pi_\iota)\pi_\iota^2 + \delta_\iota \qquad (1)$$

where λ is the `\linepenalty` (which is a constant for a single paragraph), β_ι stands for the badness of the line, π_ι represents the penalty at the line break, and δ_ι is the sum of the applicable additional demerits. Penalties lie in the range $-10000 \le \pi < 10000$ and they keep their sign in (1) although the value is squared. But the value -10000 is ignored in the calculation of the line demerits. A detailed explanation of how TEX breaks paragraphs into lines is given in Chapter 14 of [2]; Section 2 of [6] contains a shorter description that focuses on the parameters involved and the formulas for the calculations.

Traces. TEX displays most of the values that are used in formula (1) if `\tracingparagraphs` is positive. In this case the log file of the run contains trace data written by the line-breaking algorithm. Two types of lines of this trace are important to understand this article:

Break candidates signal a valid way to break a line. Their trace lines start with a single `@` and they contain the values for β_ι, π_ι, and Λ_ι. The line number ι is not yet known. For example,

```
@\par via @@1 b=1 p=50 d=7621
```
(∗)

gives $\beta_\iota = 1$, $\pi_\iota = 50$, and $\Lambda_\iota = 7621$.

Feasible breakpoints, which follow one or more break candidates, show the best way to break the text up to this point. Their trace lines start with `@@` and contain a *sequence number*, the line number, and a *fitness class*. For example, in

```
@@2: line 1.3- t=3125 -> @@0
```
(∗∗)

the sequence number is 2, the line number $\iota = 1$, and the fitness class 3. The reference `-> @@0` shows that the previous feasible breakpoint for this breakpoint has the sequence number 0.

The sequence numbers are needed to link the feasible breakpoints to determine the above mentioned set of line breaks; the sequence number 0 is used for the start of the paragraph. The fitness class is a qualitative indicator of the way the white space in the line is altered. There are four classes: *very loose*, *loose*, *decent*, and *tight* which carry the numbers 0 to 3, resp., in the trace line of type (∗∗). See pages 98–99 of [2] or Section 3 of [6] for a detailed description of the trace data.

Note that the additional demerits, δ_ι, are not listed in the trace data. They must be computed from the given values using equation (1) in the following form:

$$\delta_\iota = \Lambda_\iota - (\lambda + \beta_\iota)^2 - \mathrm{sgn}(\pi_\iota)\pi_\iota^2. \qquad (2)$$

Plain TEX sets $\lambda = 10$ [2, p. 98], so Λ_ι in (∗) contains the following amount of additional demerits:

$$\delta_\iota = 7621 - (10 + 1)^2 - +50^2 = 5000.$$

Paths. Starting with the last feasible breakpoint the line breaks can be found by going back through the sequence of the linked previous feasible breakpoints. If all feasible breakpoints appear in this sequence and all of them have only one associated break candidate, TEX has only one way to typeset the paragraph. But often several break candidates are followed by several feasible breakpoints and then many possibilities for the next breakpoint are available. TEX chooses — obeying the current setting of `\looseness` — the set of line breaks, i.e., feasible breakpoints, that has the lowest sum of line demerits, also called the *total demerits* of the paragraph. (The current total demerits at a feasible breakpoint is shown after the `t=` in lines of type (∗∗).) Each set for the whole paragraph is called a *path* and its sum of line demerits *path demerits*, Λ_p. The entirety of all paths form a *network* (see [1, Fig. 13]).

Traces can get very long when TEX finds a lot of feasible breakpoints, and then they document a lot of paths in the network. Such paths are difficult to compare when only the trace data is available. This article shows only short traces and switches to a table form for longer ones. This table form is introduced and described in Section 5 of [7]. Both TEX's trace data and the table form are also briefly explained in this article.

Additional demerits. The three additional demerit parameters fall into two groups: Two deal with hyphens at the end of lines, the third with the visual appearance of lines. The following three integer parameters hold the values of the additional demerits.

`\finalhyphendemerits` is used in the last line if the penultimate line of the paragraph ends in a hyphen.

`\doublehyphendemerits` is applied to the second line of a pair of lines if both lines end in a hyphen. But it is not applied to the last line of the paragraph.

`\adjdemerits` is charged to the second line of a pair of lines if they are *visually incompatible*. This means that the glue in the two lines is set quite differently: In one line it is extremely stretched compared to the other. (The first line of a paragraph is compared to a line with decent spacing.) In TEX's view the fitness class numbers in linked feasible breakpoints must differ by more than 1 to add the value of this parameter.

Plain TEX assigns the values 5000, 10000, and 10000 to the three parameters [2, p. 98], resp. The values of these parameters are named in this article δ_f, δ_d, and δ_a, resp. Note that δ_d and δ_f are never applied together in a line.

Implementation. In the section "More Bells and Whistles" of [1] it is written that the parameter `\adjdemerits` was added — with a high price paid in the implementation through the introduction of the fitness classes — as a more or less experimental feature. The design and implementation was done in the spring of 1980 (p. 143 of [5]); Chapter 11 in [4] shows it as entry #461. As it is available today it seems to be worth the price. Note that only the parameter `\adjdemerits` is likely to be used in the first pass as hyphenated lines must use a (typically rare) author-entered hyphen in this pass.

In the implementation of the line-breaking algorithm the values of the integer parameters that represent the additional demerits are just one summand in the calculation of the line demerits [3, §859]. But the implementation uses `\adjdemerits` in a second place to keep the number of feasible breakpoints and break candidates small [3, §836]. This means changing the values of these integer parameters might result in a different number of lines in the trace.

Of course, a path in the network created from TEX's default settings exists also as a valid set of line breaks with changed values of the parameters. But TEX does not follow a path to its end as soon as it learns that this path cannot lead to the minimal total demerits. The trace data often shows only the beginning of paths. In the table form of the trace data these partial paths are extended to a complete path using the available feasible breakpoints; such paths are described as "hidden in the trace data".

Usage. The three parameters try to prevent TEX from picking a path that contains undesirable constructions. The techniques of this article might turn out to be useful for an author to handle situations in which TEX chose unwanted line breaks. But almost always it is better to rewrite a paragraph if it looks bad than to adjust TEX's line-breaking parameters.

In the following sections the three parameters are discussed one by one. Note, however, that a change to the parameters affects all following paragraphs so that a changed value should be used inside a group around a paragraph, which must end inside this group, i.e., write an empty line or `\par` before ending the group. TEX calculates with the values assigned to the parameters that it knows at the end of a paragraph.

2 About `\finalhyphendemerits`

The effect of `\finalhyphendemerits` is probably easier to understand than the effects of the other two parameters. TEX charges δ_f to the last line of a path if and only if its penultimate line ends in a hyphen or a ligature in which the last character is a hyphen; in the `cm` fonts the line must end with a hyphen, an en-dash, or an em-dash. TEX has to look at only a single line in contrast to the other two additional demerits where TEX judges about pairs of lines.

Example 1: Description
Typeset a paragraph whose penultimate line ends in an em-dash.

TEX input

```
The em-dash at the end of this line---this
one---adds {\tt\char92finalhyphendemerits\/}
to the last line.\par
```

TEX output

The em-dash at the end of this line—this one—adds `\finalhyphendemerits` to the last line. □

(Note: The symbol '□' marks the end of an example.)

With `\tracingparagraphs` = 1 trace lines appear in the log file — here formatted to fit the column width and with line numbers for reference.

Example 1 continued: Log file contents

```
1. @firstpass
2. []\tenrm The em-dash at the end of this
   line---this one---
3. @\discretionary via @@0 b=11 p=50 d=2941
4. @@1: line 1.2- t=2941 -> @@0
5. adds \tentt \finalhyphendemerits \tenrm to
   the last line.
6. @\par via @@1 b=0 p=-10000 d=5100
7. @@2: line 2.2- t=8041 -> @@1
```

Lines 3 and 6 contain break candidates, lines 4 and 7 feasible breakpoints. As there is only one break candidate per feasible breakpoint TEX has exactly one path to typeset this paragraph. Lines 3 and 4 show a penalty for the first line; it states that the `\exhyphenpenalty` is applied, i.e., the em-dash is considered an author-entered hyphen. (Plain TEX sets `\hyphenpenalty` and `\exhyphenpenalty` to 50 [2, p. 96].) The calculation using (2) shows for this break candidate representing output line 1:

$$\delta_1 = 2941 - (10+11)^2 - 50^2 = 2941 - 441 - 2500 = 0.$$

The badness of the last line is 0; see trace lines 6 (and 7). As mentioned above a penalty of −10000 is ignored in the calculation of the line demerits. Therefore (2) gives for the second line:

$$\delta_2 = 5100 - (10+0)^2 = 5100 - 100 = 5000.$$

Thus, as expected, TEX has applied the value δ_f to the last line.

Changing this parameter. If δ_f is applied to a paragraph and if TEX's line-breaking network contains at least one path that avoids a hyphen at the end of the second-last line a high enough value for `\finalhyphendemerits` will select such a path.

Example 2: Description

Typeset a paragraph twice: First with TEX's default values and a second time with a larger value of `\finalhyphendemerits`.

TEX input

```
\noindent In this case \TeX's network shows
2 paths for its line-breaking decision.\par
```

TEX output

In this case TEX's network shows 2 paths for its line-breaking decision.

This is the trace data:

Example 2 continued: Log file contents

```
1. @firstpass
2. \tenrm In this case T[]X's network shows 2
   paths for its
3. @ via @@0 b=96 p=0 d=11236
4. @@1: line 1.1 t=11236 -> @@0
5. line-
6. @\discretionary via @@0 b=29 p=50 d=4021
7. @@2: line 1.3- t=4021 -> @@0
8. breaking decision.
9. @\par via @@1 b=0 p=-10000 d=100
10. @\par via @@2 b=0 p=-10000 d=5100
11. @@3: line 2.2- t=9121 -> @@2
```

The feasible breakpoint `@@3` has two break candidates and TEX has a choice: Go via `@@2` as suggested in `@@3` or use the break candidate in line 9 without the application of δ_f and link `@@3` to `@@1`.

To select the path that uses the feasible breakpoints `@@1` and `@@3` and thus avoids the hyphen in the penultimate line the total demerits from the path via `@@2` and `@@3`, $\Lambda_{\mathrm{p}[2,3]}$, must become larger than the path demerits of the requested path, $\Lambda_{\mathrm{p}[1,3]}$. This can be done by increasing the value of δ_f. Let's mark its new value with a prime, then the inequality

$$\Lambda_{\mathrm{p}[2,3]} + \delta'_f - \delta_f > \Lambda_{\mathrm{p}[1,3]}$$

must hold. Simple transformations give

$$\begin{aligned} & \delta'_f > \Lambda_{\mathrm{p}[1,3]} - \Lambda_{\mathrm{p}[2,3]} + \delta_f \\ \Longrightarrow \quad & \delta'_f > (11236 + 100) - 9121 + 5000 = 7215 \end{aligned}$$

and the value for `\finalhyphendemerits` should be at least 7216. The first line then becomes loose with a badness of 96 (see lines 3 and 4) instead of tight with a badness of 29 (see lines 6 and 7). It is a trade-off: To avoid the hyphen in the second-last line TEX has to select a worse solution at other places. Here it is a loose line.

Example 2 continued: TEX input

```
\finalhyphendemerits=7216
\noindent In this case \TeX's network ...
```

TEX output

In this case TEX's network shows 2 paths for its line-breaking decision.

Of course, it is not necessary to calculate the minimal δ_f; it can be set to a high value like 7500 or 10000 or 100000 to get the desired result.

3 About `\doublehyphendemerits`

The parameter `\doublehyphendemerits` is applied to the second line of a pair of lines if both lines end in a hyphen and the second line is not the last line. This restriction comes from the fact that TEX treats the end of a paragraph as if it ends in a hyphen [3, §829]. An author-entered hyphen at the end of a paragraph does not make a difference. (It is rare indeed to have a hyphen at the end of a paragraph, so it is not an important case.) Thus, in a paragraph all lines except the first and the last might be charged with δ_d, the value of this parameter.

Both explicit hyphens, which are entered by the author, and implicit hyphens, which are inserted by TEX's hyphenation algorithm, are treated in the same way by TEX when it determines if a line ends in a hyphen.

Example 3: Description
Typeset a paragraph with a lot of hyphenated lines.

TEX input

```
In this paragraph \TeX\ must apply its
hyphenation procedure. And the last line ends
in an em-dash. Parameter
{\tt\char92doublehyphendemerits\/} is
applied twice in four lines---\par
```

TEX output

In this paragraph TEX must apply its hyphenation procedure. And the last line ends in an em-dash. Parameter `\doublehyphendemerits` is applied twice in four lines—

As always, useful information is found in the trace data; the first line states that TEX uses the second pass (lines from the first pass are not shown).

Example 3 continued: Log file contents

```
1. @secondpass
2. []\tenrm In this para-graph T[]X must
    ap-ply its hy-phen-
3. @\discretionary via @@0 b=0 p=50 d=2600
4. @@1: line 1.2- t=2600 -> @@0
5. ation pro-ce-dure. And the last line ends
    in an em-
6. @\discretionary via @@1 b=7 p=50 d=12789
7. @@2: line 2.2- t=15389 -> @@1
8. dash. Pa-ram-e-ter \tentt
   \doublehyphendemerits \tenrm is ap-
9. @\discretionary via @@2 b=147 p=50 d=47149
10. @@3: line 3.0- t=62538 -> @@2
11. plied twice in four lines---
12. @\par via @@3 b=0 p=-10000 d=*
13. @@4: line 4.2- t=62538 -> @@3
```

Lines 6 and 9 show that δ_d was applied. In line 12 TEX has not calculated the demerits and sets them to 0; compare the values after `t=` in lines 10 and 13 of the trace. TEX does not calculate the demerits as the necessary feasible breakpoint at the end of the paragraph follows a possible break after the em-dash (see [3, §854]).

Remove a stack of hyphens. Note that there is no way to prevent the three consecutive hyphenated lines in example 3: TEX has no path in its line-breaking network to avoid this stack of hyphens. But if there is such a path a change of the value of `\doublehyphendemerits` can trade in higher badness values and/or more penalties and/or more visually incompatible lines and/or more lines in the paragraph to avoid such a stack of hyphens.

Table 1: Badness, penalties, and additional demerits of the line breaks for the paths of example 4

		`\par via @@` (`*` is typeset)				
`@@`	Class	$*7_0$	8_0	7_1	7_2	8_1
1	d	4_{50}	4_{50}	4_{50}	4_{50}	4_{50}
2	v				118^{a}	
3	d	0^{d}_{50}	0^{d}_{50}	0^{d}_{50}		0^{d}_{50}
4	l			51^{d}_{50}		
(4)	(d)				1^{a}_{50}	
5	d	0^{d}_{50}				0^{d}_{50}
6	t		79			
7	l	33				
(7)	(t)			15^{a}	15	
8	l		17^{a}_{50}			
(8)	(t)					98^{d}_{50}
9	d	0	0^{f}	0	0	0^{f}
# lines =		5	5	5	5	5
$\sum$ badness =		37	100	70	138	102
# a/d/f =		0/2/0	1/1/1	1/2/0	2/0/0	0/3/1
$\Lambda_{\mathrm{p}}(10)$ =		29845	41546	42242	42426	57160

Example 4: Description
Typeset a paragraph twice: First with TEX's default values and a second time with a larger value of `\doublehyphendemerits`.

TEX input

```
Final remark: Values for the badness are
sometimes stated as {\tt*} which means that it
is {\sl infinite\/} according to \TeX's rules.
For demerits such an asterisk means that the
calculation was not performed because of
certain forced conditions.\par
```

TEX output

Final remark: Values for the badness are sometimes stated as `*` which means that it is *infinite* according to TEX's rules. For demerits such an asterisk means that the calculation was not performed because of certain forced conditions.

This time the data of the trace is shown in a more compact form as there are several paths that must be studied.

Table 1 summarizes the paths that can be identified in the trace data. The table shows in the first two columns the information of the `@@`-lines, i.e., the feasible breakpoints: the sequence number and the fitness class abbreviated to the first letter of very loose, loose, decent, or tight. The next two columns present the explicitly shown `\par`-lines in the trace (like line 12 in the trace of example 3). The heading gives the sequence number after the "`via @@`" marked with a subscript 0. The next three columns are "hidden" paths in the trace that are not followed to their end by TEX; such paths get subscripts larger than 0. As these paths use `@@`-alternatives that are not shown in the trace some sequence numbers and

their fitness classes are placed in parentheses. The column head of 7_0 contains an asterisk to indicate that TEX has selected this path for typesetting.

The table entries are the badness values. A subscript signals that a penalty with the stated value occurs at the break, a superscript of 'f', 'd', or 'a' that δ_f, δ_d, or δ_a, resp., is applied.

The last four rows state the number of lines, the sum of the badness values of the path, the number of lines with δ_a, δ_d, δ_f, and the path demerits $\Lambda_{\mathrm{p}}(10)$ with the value 10 for `\linepenalty`. Most of these values are not found in the trace data; they have been computed from the information in the columns.

Column 7_0 (with the path that TEX uses) contains two applications of δ_d but the path in column 7_2 none. Therefore, it is possible to typeset the paragraph without three consecutive hyphens if the value δ_d is changed. If the new value is named δ'_d then

$$\Lambda_{\mathrm{p}[7_0]} + 2\delta'_d - 2\delta_d > \Lambda_{\mathrm{p}[7_2]}$$

must hold. This means, the path demerits of the path 7_0, which represent currently the total demerits of the paragraph, must get larger than the path demerits of path 7_2. The calculation is

$$\begin{aligned} & \delta'_d > (\Lambda_{\mathrm{p}[7_2]} - \Lambda_{\mathrm{p}[7_0]} + 2\delta_d)/2 \\ \Longrightarrow \quad & \delta'_d > (42426 - 29845 + 20000)/2 = 16290.5. \end{aligned}$$

Example 4 continued: TEX input

```
\doublehyphendemerits=16291
Final remark: Values for the badness are ...
```

TEX output

Final remark: Values for the badness are sometimes stated as `*` which means that it is *infinite* according to TEX's rules. For demerits such an asterisk means that the calculation was not performed because of certain forced conditions. ∎

The same technique can be tried to make TEX select the path of column 8_0. This time both sides list δ'_d:

$$\Lambda_{\mathrm{p}[7_0]} + 2\delta'_d - 2\delta_d > \Lambda_{\mathrm{p}[8_0]} + \delta'_d - \delta_d.$$

Therefore

$$\begin{aligned} & \delta'_d > \Lambda_{\mathrm{p}[8_0]} - \Lambda_{\mathrm{p}[7_0]} + \delta_d \\ \Longrightarrow \quad & \delta'_d > 41546 - 29845 + 10000 = 21701. \end{aligned}$$

The computed value 21702 is larger than 16291 but the latter value is sufficient to typeset path 7_2: TEX's second-best path with its default settings, path 8_0, cannot be reached from 7_0 by increasing δ_d.

Changing the other parameters too. Instead of changing only the parameter that is assigned to this kind of trouble, the other parameters can also be involved to select a specific path. To select path 7_2 in example 4 it is not necessary but in other cases it is the only way to make TEX typeset the desired path.

In the inequality to go from path 7_0 to path 7_2, only δ_d was changed. But as 7_2 applies δ_a, the inequality should better be written as

$$\Lambda_{\mathrm{p}[7_0]} + 2\delta'_d - 2\delta_d > \Lambda_{\mathrm{p}[7_2]} + 2\delta'_a - 2\delta_a.$$

This gives an inequality for the difference $\delta'_d - \delta'_a$ that must at least hold to typeset 7_2:

$$\delta'_d - \delta'_a > (\Lambda_{\mathrm{p}[7_2]} - \Lambda_{\mathrm{p}[7_0]})/2 + \delta_d - \delta_a = 6290.5.$$

That is, $\delta'_d = 10000$ and $\delta'_a = 3709$ typeset path 7_2 as well as $\delta'_d = 13300$ and $\delta'_a = 7000$.

Similar, to prefer path 8_0 to 7_0 the inequality is

$$\begin{aligned} \Lambda_{\mathrm{p}[7_0]} + 2\delta'_d - 2\delta_d > \Lambda_{\mathrm{p}[8_0]} + \delta'_a + \delta'_d + \delta'_f & \\ - \delta_a - \delta_d - \delta_f & \\ \Longrightarrow \quad \delta'_d - \delta'_a - \delta'_f > \Lambda_{\mathrm{p}[8_0]} - \Lambda_{\mathrm{p}[7_0]} + \delta_d - \delta_a - \delta_f & \\ = 6701. & \end{aligned}$$

To get from 7_2 to 8_0, i.e., to avoid that 7_2 is chosen by TEX instead of 8_0, the parameters must also fulfill

$$\begin{aligned} \Lambda_{\mathrm{p}[7_2]} + 2\delta'_a - 2\delta_a > \Lambda_{\mathrm{p}[8_0]} + \delta'_a + \delta'_d + \delta'_f & \\ - \delta_a - \delta_d - \delta_f & \\ \Longrightarrow \quad \delta'_a - \delta'_d - \delta'_f > \Lambda_{\mathrm{p}[8_0]} - \Lambda_{\mathrm{p}[7_2]} + \delta_a - \delta_d - \delta_f & \\ = -5580. & \end{aligned}$$

The two inequalities give the criteria for how to change the parameters to typeset path 8_0 instead of path 7_0. For example, $\delta'_d = 13300$, $\delta'_a = 7000$, and $\delta'_f = -750$ is one solution. (The additional demerits are integers, so they can receive negative values.)

Example 4 continued: TEX input

```
\doublehyphendemerits=13300 \adjdemerits=7000
\finalhyphendemerits=-750
Final remark: Values for the badness are ...
```

TEX output

Final remark: Values for the badness are sometimes stated as `*` which means that it is *infinite* according to TEX's rules. For demerits such an asterisk means that the calculation was not performed because of certain forced conditions. ∎

But it is much simpler to set $\delta'_f = -6702$ in order to make TEX typeset path 8_0. A single change of δ_f helps to avoid the problem with a stack of hyphens without touching δ_d or involving path 7_2. The change suggests to TEX to trade a tolerable hyphen in the penultimate line against the unwanted hyphen stack. For such a trade the integer parameter `\finalhyphendemerits` is ideal as it applies to only one line.

4 About `\adjdemerits`

The integer parameter `\adjdemerits` is applied to the second line of a pair of lines if they are visu-

ally incompatible, i.e., if their fitness classes are not adjacent in the sequence *very loose*, *loose*, *decent*, and *tight*. The first line of a paragraph is compared to the fitness class *decent*. That means that a very loose first line receives the additional demerits δ_a.

Example 5: Description
Typeset a paragraph in which each line is charged with `\adjdemerits`.

TEX input

```
So it is a v-loose line, oh, this line must be
{\sl very~loose\/} to charge the famous
``adjacent demerits'' in a {\sl first line\/}
and then follows a {\sl tight\/} line but it
forces that the very next line is {\sl loose\/}
again to charge these adjacent demerits a
third time; and then a repetition of this
tight/loose pattern makes the rest.\par
```

TEX output

So it is a v-loose line, oh, this line must be *very loose* to charge the famous "adjacent demerits" in a *first line* and then follows a *tight* line but it forces that the very next line is *loose* again to charge these adjacent demerits a third time; and then a repetition of this tight/loose pattern makes the rest. ∎

The log file contains the following trace data:

Example 5 continued: Log file contents

```
1. @firstpass
2. []\tenrm So it is a v-loose line, oh, this
   line must be
3. @ via @@0 b=100 p=0 d=22100
4. @@1: line 1.0 t=22100 -> @@0
5. \tensl very loose \tenrm to charge the
   famous ``adjacent demerits''
6. @ via @@1 b=22 p=0 d=11024
7. @@2: line 2.3 t=33124 -> @@1
8. in a \tensl first line \tenrm and then
   follows a \tensl tight \tenrm line but it
9. @ via @@2 b=37 p=0 d=12209
10. @@3: line 3.1 t=45333 -> @@2
11. forces that the very next line is \tensl
    loose \tenrm again to charge
12. @ via @@3 b=34 p=0 d=11936
13. @@4: line 4.3 t=57269 -> @@3
14. these adjacent demerits a third time; and
    then a
15. @ via @@4 b=70 p=0 d=16400
16. @@5: line 5.1 t=73669 -> @@4
17. repetition of this tight/loose pattern
    makes the rest.
18. @\par via @@5 b=16 p=-10000 d=10676
19. @@6: line 6.3- t=84345 -> @@5
```

∎

That TEX assigns to every line the value of `\adjdemerits` can happen only if the number of lines in the paragraph is even. The first line must be *very loose* and all other odd-numbered lines at least *loose*; all even-numbered lines are *tight* or *decent*. As the last line of a paragraph cannot be *loose* with the default setting of `\parfillskip` in plain TEX an odd number of lines in the paragraph is not able to charge `\adjdemerits` either to the last line or to the first if the first line is not very loose.

The trace shows that TEX has no other path to typeset the text. It has no material to trade in order to reduce the number of lines that are charged with δ_a. As there are tight lines the author can help TEX by typing a discretionary hyphen, for example, by entering `''adjacent demer\-its''`, which yields:

Example 5 continued: TEX output

So it is a v-loose line, oh, this line must be *very loose* to charge the famous "adjacent demerits" in a *first line* and then follows a *tight* line but it forces that the very next line is *loose* again to charge these adjacent demerits a third time; and then a repetition of this tight/loose pattern makes the rest. ∎

The paragraph has now a hyphenated line and is one line longer but, except for the unchanged first line, no line is charged with δ_a.

Increase the number of paths. The inserted hyphenation point shows a way that TEX might be able, without any help, to avoid at least some of the visually incompatible lines: allow hyphenation. More variation for line breaks is available if the second pass is forced. Then hyphenation with penalties and the other two additional demerits might appear.

Example 6: TEX input (Example 5 continued)

```
\pretolerance=-1 % suppress the first pass
So it is a v-loose line, oh, this ...
```

TEX output

So it is a v-loose line, oh, this line must be *very loose* to charge the famous "adjacent demerits" in a *first line* and then follows a *tight* line but it forces that the very next line is *loose* again to charge these adjacent demerits a third time; and then a repetition of this tight/loose pattern makes the rest. ∎

In the second pass the trace gets much longer so that the paths are best presented in the table form: see Table 2. The path shown in column 9_1 is used if the text is typeset in the first pass. In the second pass this path is not followed to its end by TEX; it has many more path demerits than any other set of line breaks. The path of column 11_0 represents the line breaks with the inserted discretionary hyphen.

But TEX finds a third solution. Table 2 shows that the chosen path starts with stretched white spaces in the first two lines and after a decent line the white space has to shrink in two lines. The path

Table 2: Badness, penalties, and additional demerits of the line breaks for the paths of example 6

		\par via @@ (* is typeset)				
@@	Class	9_0	$*10_0$	11_0	9_1	10_1
1	v	100^a	100^a	100^a	100^a	100^a
2	l	27_{50}	27_{50}	27_{50}		
3	t				22^a	22^a
4	d			1		
5	d	8	8			
(5)	(l)				37^a	37^a
6	l			55		
7	t	34	34		34^a	34^a
8	l			95		
9	l	70^a			70^a	
10	t		56_{50}			56_{50}
11	d			5		
12	d		0^f	0		0^f
(12)	(t)	16^a			16^a	
# lines =		6	6	7	6	6
$\sum$ badness =		255	225	283	279	249
# a/d/f =		3/0/0	1/0/1	1/0/0	6/0/0	4/0/1
$\Lambda_p(10)$ =		55305	40185	41665	84345	69225

of 11_0 typesets the whole paragraph without tight lines and its penultimate line does not end in a hyphen. Therefore in the forced second pass the setting \finalhyphendemerits $= 6481 > 41665 - 40185 + 5000$ suffices to get the path of column 11_0.

5 Summary

It is no surprise that in a paragraph with a hyphen at the end of the penultimate line a larger value of \finalhyphendemerits, δ_f, can make TEX select different line breaks which avoid this hyphen. Neither is it a surprise that a larger value of the parameter \doublehyphendemerits, δ_d, might prevent TEX from typesetting several consecutive lines that all end in a hyphen nor that a large enough value of \adjdemerits, δ_a, might reduce the number of visually incompatible lines in a paragraph.

But to change only one parameter is not always the best solution. For example, if a stack of three hyphens can be avoided by a path with two stacks of two hyphens this path gets the same amount of additional demerits as the original path if only δ_d is changed so TEX will never select this path. Only a path with a smaller number of lines charging δ_d will be eventually considered by TEX.

This article shows that it is not too difficult to determine if the problematic situation can be resolved at all: The trace data that is written to the log file if \tracingparagraphs $= 1$ must contain choices for TEX for the feasible breakpoints of the unwanted lines or the following line. On the other hand the creation of the path tables involves manual work. So these tables are not created ad hoc and are rarely used in typesetting projects.

As the value of \finalhyphendemerits is applied to only one line, a new value does not change the order induced by the path demerits for paths that apply δ_f. In this case TEX has always to select the one which has the smallest path demerits with the default value of this parameter. Therefore a low enough negative value makes TEX pick this path: In a paragraph with an unwanted construction but without a hyphenated penultimate line the setting \finalhyphendemerits $= -10000$ (or smaller) can be tried. Or first the trace could be checked as the existence of paths with an application of δ_f is readily spotted. The result might not be one of the best solutions but it is easy to produce and at least it offers TEX the ability to trade the hyphen in the second-last line with one application of δ_a or δ_d.

A forced second pass should be executed if a problem with \adjdemerits occurs in the first pass before other techniques are tried.

References

[1] Donald E. Knuth and Michael F. Plass, "Breaking paragraphs into lines", *Software — Practice and Experience* **11** (1981), 1119–1184; reprinted with an addendum as Chapter 3 in [5], 67–155.

[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, *Literate Programming*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 27, 1992.

[5] Donald E. Knuth, *Digital Typography*, Stanford, California: Center for the Study of Language and Information, CSLI Lecture Notes No. 78, 1999.

[6] Udo Wermuth, "Tracing paragraphs", *TUGboat* **37**:3 (2016), 358–373; Errata in [7], 414. tug.org/TUGboat/tb37-3/tb117wermuth.pdf

[7] Udo Wermuth, "A note on \linepenalty", *TUGboat* **38**:3 (2017), 400–414. tug.org/TUGboat/tb38-3/tb120wermuth.pdf

◇ Udo Wermuth
Dietzenbach, Germany
u dot wermuth (at) icloud dot com

Errata for a previous article. In Table $1'$ of [7] the column 12_1 should have 0^a in the row for feasible breakpoint 17 as the previous line is very loose. This adds 10000 demerits to $\Lambda_p(10)$ in this column. The values -139 and -82 in the column of 12_1 of the diagonal matrix after (18) become -207 and -164, resp.

The Treasure Chest

This is a selection of the new packages posted to CTAN (`ctan.org`) from October 2017–April 2018, with descriptions based on the announcements and edited for extreme brevity.

Entries are listed alphabetically within CTAN directories. More information about any package can be found at `ctan.org/pkg/`*pkgname*. A few entries which the editors subjectively believe to be of especially wide interest or otherwise notable are starred (*); of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the TEX community. See also `ctan.org/topic`. Comments are welcome, as always.

◇ Karl Berry
`tugboat (at) tug dot org`

biblio

`gbt7714` in `biblio/bibtex/contrib`
: Support for the Chinese bibliography standard GB/T 7714-2015.

fonts

`cmsrb` in `fonts`
: Computer Modern for Serbian and Macedonian.

`fontawesome5` in `fonts`
: Support for Font Awesome 5.

*`gfsneohellenicmath` in `fonts`
: Hellenic-style math font supporting many languages.

`morisawa` in `fonts`
: Selection of five standard Japanese fonts for pLaTeX with `dvips`.

*`plex` in `fonts`
: TEX support for the IBM Plex typeface family.

`plex-otf` in `fonts`
: (Lua,XeT)EX support for Plex as system fonts.

*`stix2-otf` in `fonts`
: OpenType Unicode STIX Two fonts.

*`stix2-type1` in `fonts`
: STIX Two in Type 1 format.

graphics

`adigraph` in `graphics/pgf/contrib`
: Augmenting directed graphs.

`graph35` in `graphics`
: Draw keys and screen items of Casio calculators.

`pgfornament-han` in `graphics/pgf/contrib`
: Traditional Chinese motifs and patterns for PGF.

`graphics/pixelart`

`pixelart` in `graphics`
: Ti*k*Z package for single-color pixel-art pictures.

`pst-antiprism` in `graphics/pstricks/contrib`
: An antiprism in PSTricks.

*`pst-calculate` in `graphics/pstricks/contrib`
: Floating point support in LaTeX, using `expl3`.

`pst-dart` in `graphics/pstricks/contrib`
: Dart boards with PSTricks.

`structmech` in `graphics/pgf/contrib`
: Ti*k*Z support for structural mechanics drawings.

`tikz-feynhand` in `graphics/pgf/contrib`
: Feynman diagrams with Ti*k*Z.

`tikz-karnaugh` in `graphics/pgf/contrib`
: PGF package for Karnaugh maps supporting many variables.

`tikz-ladder` in `graphics/pgf/contrib`
: Ladder diagrams for the PLC LD language.

`tikz-layers` in `graphics/pgf/contrib`
: Provide more graphics layers for Ti*k*Z.

`tikz-relay` in `graphics/pgf/contrib`
: Electrical diagrams with Ti*k*Z.

`tikz-sfc` in `graphics/pgf/contrib`
: Symbol collection for PLC programming sequential function chart (SFC) diagrams in Ti*k*Z.

info

`guide-latex-fr` in `info`
: Introduction to LaTeX written in French.

`latex-via-exemplos` in `info`
: LaTeX course written in Brazilian Portuguese.

*`short-math-guide` in `info`
: Guide to using `amsmath` and related packages.

language/thai

`thaispec` in `language/thai`
: Thai-language typesetting in XeLaTeX.

macros/latex/contrib

`ascmac` in `macros/latex/contrib`
: Box and picture macros with Japanese vertical writing support.

`authorarchive` in `macros/latex/contrib`
: Add self-archiving information to scientific papers.

`auto-pst-pdf-lua` in `macros/latex/contrib`
: Using LuaLaTeX together with PostScript code.

`bxtexlogo` in `macros/latex/contrib`
: Additional TEX-family logos and abbreviations.

`cascade` in `macros/latex/contrib`
: Brace constructions for math demonstrations.

`chemsec` in `macros/latex/contrib`
: Automated creation of numeric entity labels.

`colophon` in `macros/latex/contrib`
: Produce a colophon.

`crossreftools` in `macros/latex/contrib`
: Expandable extraction of `cleveref` data.

`duckuments` in `macros/latex/contrib`
: Create duckified dummy content.

`eqnnumwarn` in `macros/latex/contrib`
Warn about displaced equation numbers.
`exercisebank` in `macros/latex/contrib`
Creating, managing, reusing exercises.
`fancyhandout` in `macros/latex/contrib`
Produce nice-looking handouts.
`fduthesis` in `macros/latex/contrib`
Thesis template for Fudan University.
`footnotehyper` in `macros/latex/contrib`
Partial `hyperref` support for the `footnote` package.
`gentombow` in `macros/latex/contrib`
Generate Japanese-style crop marks.
`graphicxpsd` in `macros/latex/contrib`
Adobe Photoshop Data (PSD) support for `graphicx`.
`gridslides` in `macros/latex/contrib`
Free form slides with blocks placed on a grid.
`handin` in `macros/latex/contrib`
Lightweight template for problem sets.
`hecthese` in `macros/latex/contrib`
Thesis class for HEC Montréal.
`hulipsum` in `macros/latex/contrib`
Hungarian dummy text.
`intopdf` in `macros/latex/contrib`
Embed non-PDF files in PDF with hyperlinks.
`isopt` in `macros/latex/contrib`
Print dimension with user-defined space between number and unit.
`jkmath` in `macros/latex/contrib`
Semantic markup macros for math.
`labelschanged` in `macros/latex/contrib`
Find labels causing endless "may have changed" warnings.
`lccaps` in `macros/latex/contrib`
Lowercased (spaced) small capitals.
`llncsconf` in `macros/latex/contrib`
Extending the Springer `llncs` class.
`mathfam256` in `macros/latex/contrib`
256 math families for (u)pLaTeX and Lamed.
`mathfixs` in `macros/latex/contrib`
Spacing for fractions and roots, bold symbols, etc.
`modernposter` in `macros/latex/contrib`
Modern LaTeX poster class
`nicematrix` in `macros/latex/contrib`
Matrices with continuous dotted lines.
`nidanfloat` in `macros/latex/contrib`
Support for `[b]` placement for full-width float in double-column mode.
`outlining` in `macros/latex/contrib`
Create outlines for scientific documents.
`pdfprivacy` in `macros/latex/contrib`
Suppressing PDF metadata.
`polexpr` in `macros/latex/contrib`
Parser for polynomial expressions.
`scientific-thesis-cover` in `macros/latex/contrib`
Generic support for cover pages and affirmations.
`sectionbreak` in `macros/latex/contrib`
Support for thought breaks.
`simpleinvoice` in `macros/latex/contrib`
Easy typesetting of invoices.
`statmath` in `macros/latex/contrib`
Simple use of statistical notation.
`stealcaps` in `macros/latex/contrib`
Borrow small capitals from another font.
`textualicomma` in `macros/latex/contrib`
Using the text ',' as decimal separator in math.
`thesis-gwu` in `macros/latex/contrib`
Thesis class for George Washington University School of Engineering and Applied Science.
`timbreicmc` in `macros/latex/contrib`
Include watermarks for ICMC/USP (Brazil).
`translator` in `macros/latex/contrib`
Easy translation of strings.
`unitn-bimrep` in `macros/latex/contrib`
Class for the bimonthly PhD engineering report at Università di Trento.
`univie-ling` in `macros/latex/contrib`
Class for the applied linguistics department at Vienna University.
`xtuthesis` in `macros/latex/contrib`
XTU (China) thesis template.
`xurl` in `macros/latex/contrib`
Allow url break at any alphanumeric character.

macros/latex/contrib/beamer-contrib/themes

`beamertheme-saintpetersburg` in `m/l/c/b-c/themes`
Support for Saint Petersburg State University.

macros/latex/contrib/biblatex-contrib

`biblatex-ext` in `m/l/c/biblatex-contrib`
Extended BibLaTeX standard styles.

macros/luatex/latex

`bezierplot` in `macros/luatex/latex`
Approximate smooth function graphs with splines.
`gurps` in `macros/luatex/latex`
Generic Universal Role Playing System materials.
`plantuml` in `macros/luatex/latex`
Rendering diagrams specified in PlantUML.
`typewriter` in `macros/luatex/latex`
Typeset with randomly variable monospace font.
`wallcalendar` in `macros/luatex/latex`
Wall calendar class with custom layouts and internationalization.

macros/xetex/latex

`sexam` in `macros/xetex/latex`
Support for Arabic `exam` scripts.

support

`ctan-o-mat` in `support`
Upload or validate a package for CTAN.
`lyluatex` in `support`
Include LilyPond scores in a LuaLaTeX document.
`npp-for-context` in `support`
ConTeXt plugin for Notepad++.

Die TEXnische Komödie 4/2017–1/2018

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`dante.de`). (Non-technical items are omitted.)

Die TEXnische Komödie 4/2017

Die TEXnische Komödie 1/2018

[Received from Herbert Voß.]

Zpravodaj 2017/1–4

Editor's note: *Zpravodaj* is the journal of CSTUG, the TEX user group oriented mainly but not entirely to the Czech and Slovak languages (`cstug.cz`).

Zpravodaj 2017/1–2

Petr Sojka, Úvodník [Introduction]; pp. 1–2

This editorial discusses CSTUG's new publishing policy and comments on this issue's articles. Go forth and participate in CSTUG to make the bright future of TEX & Friends a reality! *You can!*

Hans Hagen, ConTEXt–Lua Documents; pp. 3–54

In ConTEXt, it is now possible to prepare documents in a mixture of TEX, XML, MetaPost, and Lua. The article gives a short introduction into the programming language of Lua and then goes on to describe how Lua can be used for programming in ConTEXt MkIV. 10.5300/2017-1-2/3

Hans Hagen, Exporting XML and ePub from ConTEXt; pp. 55–63

The article describes the XML backend of ConTEXt, which can be used to produce structured XML documents out of a TEX input. One of the many applications of the XML backend is the conversion to ePub e-book format, which the article covers in detail. 10.5300/2017-1-2/55

Hans Hagen and Idris Samawi Hamid, Oriental TEX: Optimizing paragraphs; pp. 64–97

The article describes the state of the art in paragraph optimization for Arabic as implemented in ConTEXt. The implementation is introduced using Latin script examples. The article proceeds to describe the main features of Arabic script and known approaches towards paragraph optimization. One of the described approaches is then implemented and used to typeset a passage from the Qur'ān. 10.5300/2017-1-2/64

Taco Hoekwater, MetaPost: PNG output; pp. 98–100

[Printed in *TUGboat* **34**:2.] 10.5300/2017-1-2/98

Aleš Kozubík, Mapy v LATEXových dokumentoch – predstavenie balička `getmap` [Maps in LATEX documents — an introduction of the `getmap` package]; pp. 101–109

The aim of this article is to introduce the `getmap` package. This package supports inclusion into LATEX documents the map materials obtained from the external resources such as OpenStreetMap and Google Maps, possibly even with the support of Google Street View. In the simplest case, the specification of an address is sufficient. The package loads the map using the `\write18` feature, which must be activated to use this package. The image will be downloaded by an external Lua script that can be used also from the command line. 10.5300/2017-1-2/101

Vít Novotny, Konference TUG@BachoTEX 2017 [Conference TUG@BachoTEX 2017]; pp. 110–116

The article is a summary of the TUG@BachoTEX 2017 conference, which was held from April 29 to May 3 jointly by GUST and TUG at Bachotek near Brodnica, Poland. 10.5300/2017-1-2/110

Zpravodaj 2017/3–4

Petr Sojka, Úvodník [Introduction]; p. 117

This editorial again discusses CSTUG's new publishing policy and comments on this issue's articles.

Petr Sojka, Vít Novotny, TEX na školách? Samozřejmě ano! Příklad užití TEXu ma Fakultě informatiky Masarykovy univerzity [TEX in Schools? Just Say Yes: The use of TEX at the Faculty of Informatics, Masaryk University]; pp. 118–137

[Printed in *TUGboat* **38**:2.] 10.5300/2017-3-4/118

Hans Hagen, Lua in MetaPost; pp. 138–154

In LuaTEX, it is now possible to run snippets of Lua code from within MetaPost. The article describes the mechanism, the low-level interface available in LuaTEX, as well as the high-level interface available in ConTEXt through example. 10.5300/2017-3-4/138

Peter Wilson, Mělo by to fungovat VI – Odstavce [It might work. VI — Paragraphs]; pp. 155–164

This paper demonstrates possibilities of setting the TEX paragraph parameters for different, even less usual, paragraph justifications. [Printed in *TUGboat* **28**:2.] 10.5300/2017-3-4/155

TUG financial statements for 2017

Karl Berry, TUG treasurer

The financial statements for 2017 have been reviewed by the TUG board but have not been audited. As a US tax-exempt organization, TUG's annual information returns are publicly available on our web site: `https://tug.org/tax-exempt`.

Revenue (income) highlights

Membership dues revenue was down about $10,000 in 2017 compared to 2016, in large part due to doubling the electronic discount. Contributions were down about $2,500. Product sales (mainly Lucida) was up about $1,300. The annual conference was not on our budget in 2017. Other categories were about the same. Overall, 2017 income was down 11%.

Cost of Goods Sold and Expenses highlights, and the bottom line

Postage and delivery costs were up about $1,600, due to increasing postal costs worldwide. Other cost categories were about the same. No unusual fees were incurred in 2017.

The bottom line for 2017 was strongly negative, about $15,200.

Balance sheet highlights

TUG's end-of-year asset total is down by around $9,600 (4.9%) in 2017 compared to 2016, following the bottom-line loss.

Committed Funds are reserved for designated projects: LaTeX, CTAN, the TeX development fund, and others (`https://tug.org/donate`). Incoming donations are allocated accordingly and disbursed as the projects progress. TUG charges no overhead for administering these funds.

The Prepaid Member Income category is member dues that were paid in earlier years for the current year (and beyond). The 2017 portion of this liability was converted into regular Membership Dues in January of 2017. The payroll liabilities are for 2017 state and federal taxes due January 15, 2018.

Summary

We ended 2017 with 15 fewer members than in 2016. In 2018, TUG started a trial membership initiative which has attracted some new members. If the new trial members renew as regular members, we can hope the downward membership trend will slow or even stop.

Additional ideas for attracting members, or benefits TUG could provide, would be very welcome.

◇ Karl Berry, TUG treasurer
`https://tug.org/tax-exempt`

TUG 12/31/2017 (vs. 2016) Revenue, Expense

	Jan - Dec 17	Jan - Dec 16
Ordinary Income/Expense		
Income		
Membership Dues	76,502	86,460
Product Sales	7,100	5,801
Contributions Income	7,654	10,681
Annual Conference		-699
Interest Income	546	575
Advertising Income	315	315
Services Income	761	1,176
Total Income	92,878	104,309
Cost of Goods Sold		
TUGboat Prod/Mailing	23,677	24,896
Software Production/Mailing	2,599	2,479
Postage/Delivery - Members	2,901	1,356
Lucida Sales Accrual B&H	2,895	2,263
Member Renewal	364	384
Total COGS	32,436	31,378
Gross Profit	60,442	72,931
Expense		
Contributions made by TUG	2,000	2,000
Office Overhead	13,741	14,934
Payroll Exp	63,186	63,167
Professional Fees	38	13,878
Interest Expense	45	50
Total Expense	79,010	94,029
Net Ordinary Income	-18,568	-21,098
Other Income/Expense		
Other Income		
Prior year adjust	3,356	-1
Net Other Income	3,356	-1
Net Income	**-15,212**	**-21,099**

TUG 12/31/2017 (vs. 2016) Balance Sheet

	Dec 31, 17	Dec 31, 16
ASSETS		
Current Assets		
Total Checking/Savings	184,765	193,913
Accounts Receivable	275	715
Total Current Assets	185,040	194,628
TOTAL ASSETS	**185,040**	**194,628**
LIABILITIES & EQUITY		
Liabilities		
Committed Funds	42,972	35,842
TUG conference	596	
Prepaid member income	6,070	6,850
Payroll Liabilities	1,080	1,083
Total Current Liabilities	53,416	47,792
TOTAL LIABILITIES	**53,416**	**47,792**
Equity		
Unrestricted	146,836	167,934
Net Income	-15,212	-21,098
Total Equity	131,624	146,836
TOTAL LIABILITIES & EQUITY	**185,040**	**194,628**

TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `tug.org/consultants.html`. If you'd like to be listed, please see there.

Aicart Martinez, Mercè
Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) ono.com`
Web: `http://www.edilatex.com`
We provide, at reasonable low cost, LaTeX or TeX page layout and typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990. For more information visit our web site.

Dangerous Curve
+1 213-617-8483
Email: `typesetting (at) dangerouscurve.org`
We are your macro specialists for TeX or LaTeX fine typography specs beyond those of the average LaTeX macro package. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical TeX and LaTeX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a TeX book.

Dominici, Massimiliano
Email: `info (at) typotexnica.it`
Web: `http://www.typotexnica.it`
Our skills: layout of books, journals, articles; creation of LaTeX classes and packages; graphic design; conversion between different formats of documents.

We offer our services (related to publishing in Mathematics, Physics and Humanities) for documents in Italian, English, or French. Let us know the work plan and details; we will find a customized solution. Please check our website and/or send us email for further details.

Hendrickson, Amy
57 Longwood Ave. #8
Brookline, MA 02446
+1 617-738-8029
Email: `amyh (at) texnology.com`
Web: `http://texnology.com`
LaTeX Macro Writing: Packages for print and e-publishing; Sophisticated documentation for users. Book and journal packages distributed on-line to thousands of authors.

More than 30 years' experience, for major publishing companies, scientific organizations, leading universities, and international clients.

Graphic design; Software documentation; LaTeX used for Data Visualization, and automated report generation; e-publishing, design and implementation; Innovation to match your needs and ideas.

LaTeX training, customized to your needs, on-site — have taught classes widely in the US, and in the Netherlands and Sweden.

See the TeXnology website for examples. Call or send email: I'll be glad to discuss your project with you.

Latchman, David
2005 Eye St. Suite #6
Bakersfield, CA 93301
+1 518-951-8786
Email: `david.latchman (at) texnical-designs.com`
Web: `http://www.texnical-designs.com`
LaTeX consultant specializing in the typesetting of books, manuscripts, articles, Word document conversions as well as creating the customized packages to meet your needs. Call or email to discuss your project or visit my website for further details.

Peter, Steve
+1 732 306-6309
Email: `speter (at) mac.com`
Specializing in foreign language, multilingual, linguistic, and technical typesetting using most flavors of TeX, I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. In addition, I've helped publishers write, maintain, and streamline TeX-based publishing systems. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Sofka, Michael
8 Providence St.
Albany, NY 12203
+1 518 331-3457
Email: `michael.sofka (at) gmail.com`
Personalized, professional TeX and LaTeX consulting and programming services.

I offer 30 years of experience in programming, macro writing, and typesetting books, articles, newsletters, and theses in TeX and LaTeX: Automated document

Sofka, Michael (cont'd)
conversion; Programming in Perl, C, C++ and other languages; Writing and customizing macro packages in TEX or LATEX, `knitr`.

If you have a specialized TEX or LATEX need, or if you are looking for the solution to your typographic problems, contact me. I will be happy to discuss your project.

TEXtnik
Spain
Email: `textnik.typesetting (at) gmail.com`
Do you need personalised LATEX class or package creation? Maybe help to finalise your current typesetting project? Any problems compiling your current files or converting from other formats to LATEX? We offer +15 years of experience as advanced LATEX user and programmer. Our experience with other programming languages (scripting, Python and others) allows building systems for automatic typesetting, integration with databases, ... We can manage scientific projects (Physics, Mathematics, ...) in languages such as Spanish, English, German and Basque.

Veytsman, Boris
132 Warbler Ln.
Brisbane, CA 94005
+1 703 915-2406
Email: `borisv (at) lk.net`
Web: `http://www.borisv.lk.net`
TEX and LATEX consulting, training, typesetting and seminars. Integration with databases, automated

Veytsman, Boris (cont'd)
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.

Webley, Jonathan
Flat 11, 10 Mavisbank Gardens
Glasgow, G1 1HG, UK
07914344479
Email: `jonathan.webley (at) gmail.com`
I'm a proofreader, copy-editor, and LATEX typesetter. I specialize in math, physics, and IT. However, I'm comfortable with most other science, engineering and technical material and I'm willing to undertake most LATEX work. I'm good with equations and tricky tables, and converting a Word document to LATEX. I've done hundreds of papers for journals over the years. Samples of work can be supplied on request.

Production notes

Karl Berry

In the previous issue, Barbara's editorial (`tug.org/TUGboat/tb38-3/tb120beet.pdf`) mentioned the plan to create "complete" PDFs for all issues. This has been done, to the extent presently practical, and they are linked from the individual issues' contents pages under `tug.org/TUGboat/Contents`. Cover pages, ancillary items, and assorted other refinements to the contents and "lists" files (authors, titles, keywords, also linked on the `Contents` page) were made in the course of the work.

The results are far from perfect, since the sources for many older issues are incomplete or missing, ditto the files sent to the printer, ditto the scanning done over the years. Sometimes the only files available had cropmarks and printer's instructions. Nevertheless, I felt it more useful to include everything available, for the sake of the content, rather than omit material for its imperfections.

Going forward, the plan is for the page numbers in both the main and by-difficulty tables of contents to be active links in the complete PDF to the corresponding page. This was done for the first time in the last issue, 38:3. (There is no plan to do this for older issues.)

On the TEXnical side, I used this pdfTEX primitive command to create these links:

```
\pdfstartlink
  attr{/Border [0 0 0]}
  goto page \tublinkdestpage{/XYZ}\relax
```

where: 1) `attr{/Border...}` prevents boxes from being displayed around the links by PDF readers, which I find remarkably distracting; 2) `gotopage.../XYZ` specifies the destination page and to preserve the current position when moving, instead of zooming the display in or out; 3) `\tublinkdestpage` was previously defined as a counter, set to the corresponding "physical" PDF page number (always starts at 1).

Thus, making the actual link is reasonably straightforward. Getting to that point, though, was more painful: disentangling the longstanding *TUGboat* cover machinery, which previously made several of the cover pages at once, with DVI output (dating back to the days when *TUGboat* paste-up involved scissors and glue, not just moving bytes around). I wanted to switch to pdfTEX directly for the sake of making the link (as opposed to trying to pass links through DVI to PostScript to PDF), and to process each page in a separate run, due to macro dependencies and just for my sanity. The final result is not intended to be visibly different.

One other (unrelated) improvement I learned of recently was to run the complete PDF through Ghostscript using `-dCompressFonts=true`, which compresses our Type 1 fonts into Type 1C, usually making the final file significantly smaller.

Happy *TUGboat* browsing!

◇ Karl Berry
`tug.org/TUGboat`

The 39th Annual Meeting of the TEX Users Group

July 20–22, 2018

Institute for Pure and Applied Mathematics — IMPA
Rio de Janeiro, Brazil

A SATELLITE OF

ICM 2018 RIO DE JANEIRO INTERNATIONAL CONGRESS OF MATHEMATICIANS

Official satellite conference of
International Congress of Mathematicians 2018

tug2018@tug.org
http://tug.org/tug2018

Sponsored by the TEX Users Group, DANTE e.V., and ICM'18.

Calendar

2018

Apr 4 – 6 DANTE 2018 Frühjahrstagung and 58th meeting, Passau, Germany. www.dante.de/events.html

Apr 12 – 14 TYPO Labs 2018, "How far can we go?", Berlin, Germany. typotalks.com/labs

Apr 28 – May 2 BachoTEX 2018, "What's to stay, what's to go", 26th BachoTEX Conference, Bachotek, Poland. www.gust.org.pl/bachotex

May 1 **TUG 2018** and **Practical TEX 2018**. Deadlines: early bird registration, abstracts for presentation proposals, tug.org/tug2018 and practicaltex2018

May 17 – 19 TYPO Berlin 2018, "Trigger", Berlin, Germany. typotalks.com/berlin

Jun 3 – 8 17th Annual Book History Workshop, Texas A & M University, College Station, Texas. library.tamu.edu/book-history

Jun 4 – 15 Mills College Summer Institute for Book and Print Technologies, Oakland, California. millsbookartsummer.org

Jun 14 – 16 /gʁafematik/: Graphemics in the 21st century — From graphemes to knowledge, IMT Atlantique, Brest, France. conferences.telecom-bretagne.eu/grafematik/

Jun 24 – 30 Digital Humanities 2018, Alliance of Digital Humanities Organizations, El Colegio de México and Universidad Nacional Autónoma de México (UNAM), Mexico City. adho.org/conference

Jun 25 – 27 **Practical TEX 2018**, Rensselaer Polytechnic Institute, Troy, New York. tug.org/practicaltex2018

Jun 25 – 29 SHARP 2018, "From First to Last: Texts, Creators, Readers, Agents". Society for the History of Authorship, Reading & Publishing. Sydney, Australia. www.sharpweb.org/main

Jun 28 – 29 Centre for Printing History & Culture, "Script, print, and letterforms in global contexts: the visual and the material", Birmingham City University, Birmingham, UK. www.cphc.org.uk/events

Jul 1 **TUG 2018** deadline for preprints for printed program. tug.org/tug2018

Jul 5 – 7 Sixteenth International Conference on New Directions in the Humanities (formerly Books, Publishing, and Libraries), "Reconsidering Freedom", University of Pennsylvania, Philadelphia, USA. thehumanities.com/2018-conference

---

TUG 2018 (satellite conference to the International Congress of Mathematicians) **Rio de Janeiro, Brazil.**

Jul 20 – 22 The 39th annual meeting of the TEX Users Group. tug.org/tug2018

Jul 30 – Aug 3 Balisage: The Markup Conference, Rockville, Maryland. www.balisage.net

Aug 1 – 5 TypeCon 2018, 20th anniversary, Portland, Oregon. typecon.com

Aug 4 *TUGboat* **39**:2 (proceedings issue), submission deadline.

Aug 12 – 16 SIGGRAPH 2018, "Generations", Vancouver, Canada. s2018.siggraph.org

Sep 2 – 8 12th International ConTEXt Meeting, "Unusual usage of ConTEXt", Prague–Sibřina, Czech Republic. meeting.contextgarden.net/2018

Sep 9 – 14 XML Summer School, St Edmund Hall, Oxford University, Oxford, UK. xmlsummerschool.com

Sep 29 *TUGboat* **39**:3, submission deadline.

Oct 5 – 7 Oak Knoll Fest XX, New Castle, Delaware. www.oakknoll.com/fest

Status as of 15 March 2018

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 lists.tug.org/tex-meetings. Interested users can subscribe and/or post to the list, and are encouraged to do so.

Other calendars of typographic interest are linked from tug.org/calendar.html.

TUGBOAT Volume 39 (2018), No. 1